Design Milestone

Eric Ruleman (eruleman), Nicholas Mizoguchi (nickmm), Victor Horta (vhorta)

1. Describe your strategy for using ANTLR to create your AST. You should also describe how you will handle errors in the input file.

- ANTLR Strategy

We are going to walk through the tree (using a Depth First Search algorithm), and as we visit the Concrete Tree nodes, we will feed two separated lists: one of “raw notes”, one of lyrics. We will also extract the data we need from the header in order to create the correct notes (such as correct default duration, correct key).

After we have traversed the entire tree and applied all the corrective methods (converting our raw notes into the correct tuning and correct length), we should then be left with a list of notes and a list of lyrics.

One important thing to highlight is that, since the Java MIDI library does not require rests, we will only use ‘rests’ in order to determine when the next note in a list of notes start.

- Handling errors in the input file

Our grammar is strictly accepting only well-formatted '.abc' files.

For instance: sometimes, ‘bag’ can mean a sequence of notes (‘b’, ‘a’ and ‘g’), or it can just be a valid text input (as a name of a tune). To be able to handle these specific cases, we tried to create small meaningful tokens, and group them according to their behavior. Inputs like ‘m’ or ‘b’ can have multiple meanings, and they were tokenized separately. This way, we could create a completely functional, but still organized way to build our Abstract Syntax Tree using ANTLR. This is also ready for change, as we could improve the expressivity of our grammar by adding similar rules following the same strategy.

In order to create a fully functional MIDI file, together with its own lyrics, our lexer handles all the possible valid cases, so that if the user offers an '.abc' file with any kind of flaws (e.g.: inserting invalid characters as notes, or not offering the demanded header descriptions), it simply won’t work. We build our lexer and parser rules to be able to FAIL FAST! At the time the ANTLR faces an error, it throws a RuntimeException, which could later be handled to give the final user a more informative response.

One special invalid input that our program will accept is when a measure is malformed. If there are less than or greater than the number of beats required per measure, we will ignore bar lines when building the AST. As a result, malformed measures will be accepted.

2. Describe how you will take a representation of the input (e.g., your AST) and transform it into a format that you can cleanly play using SequencePlayer.

An ABCMusic instance is built using the concrete tree given by the Grammar. An ABCMusic instance contains the beatsPerMinute and ticksPerBeat necessary to instantiate a SequencePlayer. ABCMusic gives us a list of Notes, which has the necessary information (Pitch, startTick, and duration) to create each MIDI note in the SequencePlayer. Also, ABCMusic gives a list of Syllables, which has each syllable that will be shown on screen, and also the tick when it should be shown.

3. List the components of your system that you believe can and should be tested. For each of these components, describe your testing strategy and describe at least three specific test cases you plan to have.

Lexer

- Test header, single note, multiple notes, duplets, triplets, quadruplets, octaves, and accidentals, chord, lyrics, and voices.
- Lexe all six sample input files.

Parser

- Parse all six sample input files.
- Testing invalid inputs:
    - Do not accept invalid headers: Must include X: T: K:, and each header must be on its own line.
    - Do not accept non-notes in lines of music (i.e. H).
    - Do not accept a tune with no notes.

ABCMusic.getListOfNotes()

- Compare parser results with manually transcribed notes. (3 tests)
- Check if ABCMusic is immutable by ensuring that a copy of the listOfNotes is returned.

ABCMusic.getLyrics()

- Compare parser results with manually transcribed lyrics. (3 tests)
- Check if ABCMusic is immutable by ensuring that a copy of the lyrics is returned.

ABCMusic.getBeatsPerMinute()

- Compare parser results with manually transcribed beatsPerMinute (3 tests).

ABCMusic.getTicksPerBeat()

- Compare parser results with manually transcribed beatsPerMinute (3 tests).
- Ensure that shortest note of the piece can be played.

ABCMusic.getKey()

- Check if the key was correctly applied by comparing parser results with manually transcribed notes. (3 tests)

Syllable

- Check text and tick attributes.
- Check if Syllable is immutable.
- Check if special characters are being treated correctly (e.g.: a tilde keeps two strings together as a single lyrics syllable; the underscore must come at the end of a syllable).

Note

- Check Pitch, duration, and startTick attributes.
- Check if Note is immutable.

4. Datatypes

Here are the datatypes required by our music player.

Main:

```
  - Methods:
      // Calls the play() method.
      main()


      // Plays the file.
      play(String file)
```

ABCMusic: ***TuneData***

```
// ABCMusic contains the title, composer, and key of a tune; the
//    ticksperPerBeat and beatsPerMinute used to construct a
//    SequencePlayer; and the the listOfNotes and lyrics.
//    (immutable)
 -Attributes:
   - title: String,
   - composer: String,
   - key: String, // NOTE: This is for the user's convenience. The
           // transposition of the notes to the correct key occurs
           // during the traversal of the AST.
   - listOfNotes(List<Note>),
   - lyrics(List<Syllable>),
   - ticksPerBeat: integer,
   - beatsPerMinute: integer,


 -Constructor
ABCMusic(String title, String composer, String key, List<Note> listOfNotes,
List<Syllable> lyrics, int ticksPerBeat, int beatsPerMinute){
      // TODO: implement the constructor method of ABCMusic
}


 -Methods:
      // Return a copy of the lyrics (in order to preserve
      //    immutability of ABCMusic)
   + getLyrics()
      // Return a copy of the ListOfNotes (in order to preserve
      //    immutability of ABCMusic)
   + getListOfNotes()
      // Return ticksPerBeat
   + getTicksPerBeat()
      // Return beatsPerMinute
   + getBeatsPerMinute()
```

Note(pitch: Pitch, duration: int, startTick: int)
// Immutable. A Note is a pitch with a specified duration and
// startTick.

-Attributes:
- pitch: Pitch
- duration: int
- startTick: int

-Methods:
// Returns Pitch.toMidiNote()
+ getMidiNote()
// Returns duration
+ getDuration()
// Returns getStartTick
+ getStartTick()

Syllable(text: String, tick: int)
// Immutable. Represents the smallest unit of a lyric. Contains the //
syllable to be sung as well as the tick to be displayed on.

-Attributes:
- text: String,
- tick: int

-Methods:
// Returns the syllable to be sung
+ getText()
// Returns the syllable tick to be displayed on
+ getTick()

// Representation of pitch of a note.
// @value integer value that represents a note. Its value is defined by
scale[]
// @param accidental represents the transpose value of the accident
// @param value

```
*** ALREADY IMPLEMENTED. FOR REFERENCE ONLY. ***
*** Check Pitch.java for documentation       ***
Pitch(value:int, accidental:int, octave: int)
Attributes:
      - value : int
      - accidental : int
      - octave : int
      - static scale : int[]


Methods:
      + accidentalTranspose(int)
      + octaveTranspose(int)
      + transpose(int)
      + difference(Pitch)
      + toMidiNote()
      + lessThan(Pitch)
```

Once we have correctly implemented the ABCMusic and the Main classes, we will implement KaraokePlayer as follows:

KaraokePlayer:

```
(mutable)
 -Attributes:
   - listOfTunes(List<ABCMusic>)
   - listener(LyricListener)


 -Methods:
   // From a File, create a ABCMusic and add it to the end of
   //   listOfTunes
   + addTune(filePath: File)

   // Removes the ABCMusic from the indicated position in the
   //       listOfTunes.
   // Returns true if the remove was successful, or false otherwise.
   + removeTune(position: int)

   // Play next tune in listOfTunes (if listOfTunes.length() > 0).
   // Will query the ABCMusic for ticksPerBeat and beatsPerMinute in
   // order to instantiate a SequencePlayer. Will then query
   // ABCMusic
   // for listOfNotes and lyrics, and then "load" these values into
   // the SequencePlayer. Will then call SequencePlayer.play().
   + playNextTune()


   // Prints the title and composer of the next tune in listOfTunes
   + showNextTune()
```