Homework # 1

due Monday, September 11, 10:00 PM

In this assignment, you will implement an ADT for a music playing program. We provide the driver and JUnit test cases. You need to implement the Note ADT. All methods marked in this document with an asterisk are provided for you.

1 Musical Notation and MIDI

Music uses its own notation. See for example this excerpt from the tune for “Somewhere over the Rainbow.”

\[
\begin{align*}
\text{C5} & \times 0.5 \\
\text{C6} & \times 0.5 \\
\text{B5} & \times 0.25 \\
\text{G5} & \times 0.125 \\
\text{A5} & \times 0.125 \\
\end{align*}
\]

You do not need to understand this notation for CompSci 351, but in case you are interested, here’s a quick introduction: The first note is a middle-C half-note (open head with a tail); the second note is another half-note, an octave higher (also a C). The third note (after the bar which simply marks time) is a quarter note (filled head). The fourth and fifth notes are eighth notes connected with a beam. For more information, search “Musical Notation” for a page such as “The Method Behind The Music”.

Every octave (doubling of frequency), the pitch names repeat. An octave above middle C is another C. In order to distinguish them, we use subscripts such as C_5, which in the MIDI specification refers to middle C. The next highest C is C_6, and so on. If you include “chromatic” notes, every octave contains 12 pitches: C, C♯, D, D♯, E, F, F♯, G, G♯, A, A♯, B. The “♯” symbol is pronounced “sharp.” There is no E♭ or B♯ because E and F are already very close, as are B and C. Musical notation permit one to write (say) D♭ as a synonym for C♯, but for our class we only use sharps. We write the names of pitches with lowercase letters, and when we write a note as a string, we put an “x” between the pitch and the duration. Thus the five notes of the song can be written:

\[
\begin{align*}
\text{c5} & \times 0.5 \\
\text{c6} & \times 0.5 \\
\text{b5} & \times 0.25 \\
\text{g5} & \times 0.125 \\
\text{a5} & \times 0.125 \\
\end{align*}
\]

Recall that 0.125 is \(\frac{1}{8}\) in decimal form, and that the fourth and fifth notes in our example are eighth notes. Unlike the traditional musical notation above, you will need to understand this format.

When we work with notes in our program, we convert the strings into a form which gives the pitch and duration separately. It will also convert the pitch name (such as \text{c#6}) into a “MIDI” value. The MIDI standard has a lot of details for representing music. Your task will only need concern itself with MIDI pitch values. MIDI represents pitches with integers, with C_0 = 0, and increasing by one for each pitch up to G_{10} = 127:
We restrict duration to be no less than one sixty-fourth note, and no greater than eight whole notes, although apparently in modern notation there is nothing larger than a double whole note (“breve”). Our class also supports “rests,” in which no sound is made for a particular duration. We use a special, otherwise illegal, pitch (128) for rests.

2 Note

A Note represents a particular pitch and duration. The Note class is a simple immutable class with three fields. By “immutable,” we mean that the fields are declared final. Since the fields cannot be modified after construction, the instances of the class are essentially frozen (“immutable”). Immutable objects are easy to reason about since they don’t change their values. In later homeworks, we will deal with mutable objects and then we need to worry about “aliases,” with which a mutation performed in one context will cause (surprising) changes observable in a different context.

The Note class provides the following operations:

Note(String, double) The main constructor in which the user provides the pitch name and the duration.

Note(int, double) An alternate constructor using a MIDI value for the pitch.

*Note(String) An alternate constructor using the (whole) string representation of a note.

int toMidi(String) Convert a pitch name into its MIDI value. (e.g. c#6 → 73) (static)

String toPitch(int) Convert a MIDI value into its pitch name. (e.g. 41 → f3) (static)

*String getPitch() Returns the name of this note’s pitch.

*int getMidiPitch() Returns the MIDI value of this note’s pitch.

*double getDuration() Returns the duration of this note.

Note stretch(double) Return a new note with the same pitch but with the duration equal to the product of this note’s duration with the argument.

Note transpose(int) Return a new note transposed higher (or lower, if the argument is negative) by the given interval (number of MIDI values).

In C++, this would be done by declaring them const, not that we’ll be using C++ this semester.

*Provided for you.
String toString() Returns a string of the form “p x d” where p is the name of the pitch and d is the duration.

boolean equals(Object) Return true if the argument is a note with the same pitch and duration.

int hashCode() Returns an int associated with this Note. You should compute a unique integer efficiently (without computing string hashcodes).

Recall that equals, hashCode, and toString are overridden from Object.

3 MusicBox driver

We provide a driver and two sample songs to test your ADT with. It loads in a series of notes and plays them on a given instrument.

4 Files

We will be using “git” to access and turn in programs. Follow the instructions given in the lab to clone your homework repository onto your own or lab computer. If you want to get started sooner than that, you can access it from the URI, by SSH from andrew.cs.uwm.edu (port 53211), file path

/afs/cs.uwm.edu/users/classes/cs351/401/ePanther_.git/homework1.git

where _ePanther_ is your login id.

Make sure you always push changes back (commit is insufficient) before switching computers and before the deadline. It is a good idea to commit (and push) regularly while working, so that your progress is preserved even if something goes wrong.

The repository for this assignment includes the following files:

src/UnlockTest.java Unlock the tests without running them. It is highly recommended you run this before beginning any implementation.

src/TestNote.java Tests you can run against your implementation.

src/edu/uwm/cs351/MusicBox.java The driver.

src/edu/uwm/cs351/Note.java The skeletonized Note class.

songs/ A folder with two sample songs to provide as program arguments to MusicBox.java.

Your task is to complete src/edu/uwm/cs351/Note.java

This file must exist (be updated!) in your homework1.git repository on andrew.cs.uwm.edu before the deadline (10:00 PM, Monday).