6.005 Elements of Software Construction
Fall 2008

6.005 Elements of Software Construction
Fall 2008
Lab 2: abc Music Player

In this lab, you will familiarize yourself with the abc notation and the Java MIDI API as a preparation for Project 2. You will also get a chance to practice using the Visitor pattern.

Before Lab

Before coming to lab, please do the following:

- Read this hand out and the Project 2 description.
- Basic knowledge about the Western musical notation will be helpful in this lab as well as the project. Skim through the Wikipedia article on modern musical symbols.
- For exercises on abc (and for debugging your implementation in Project 2), you may also find existing abc players useful. There are many players available on different platforms. Here are some of the recommended ones:
  - **Linux**: abcMIDI
  - **Mac**: BarFly
  - **Windows**: ABCexplorer
  - **Web-based**: convert-a-matic

  More abc applications can be found here.
- Check out the `abcPlayer` project from your group SVN repository. The repository is located at

  `svn+ssh://athena.dialup.mit.edu/mit/6.005/svn/groups/abcPlayer/group`*K*

  where *K* is your assigned group number.

Java MIDI Sequencer

In the first section of the lab, you will learn about the Java MIDI Sequencer, which allows you to schedule a series of notes to be played at certain time intervals.

Look at the package sound under `src`, and study the provided class, `SequencePlayer`. For this project, you will not need to modify this class, but you should become comfortable using its

constructor and the two methods, `addNote` and `play`.

addNote(int note, int startTick, int numTicks): This method schedules a note to be played at `startTick` for the duration of `numTicks`. Here, a "tick" is similar to a time step. At the beginning of a musical piece, the global tick is initialized to 0, and as the music progresses through the notes, the global tick is incremented by some number.

The first parameter `note` is a frequency value that corresponds to the pitch of a note. The provided class `Pitch` contains a number of useful methods for working with pitches. The method `toMidiFrequency` returns the frequency value of the particular note, and `transpose` can be used to transpose the note some number of semitones up or down.

SequencePlayer(int beatsPerMinute, int ticksPerQuarterNote): The constructor for `SequencePlayer` takes two parameters:

- The first, `beatsPerMinute`, is the tempo of a musical piece, expressed in the number of beats per minute (BPM), where each beat is equal to the duration of **one quarter note**. The BPM to be used for a particular piece depends on the value of the optional tempo field ('Q') in the input abc file. When this field is absent, the default tempo is 100 BPM, where each beat is equal in duration to the **default note length** (indicated by the field 'L').

- The second parameter, `ticksPerQuarterNote`, corresponds to the number of ticks per quarter note. Note that ticks used by the sequencer are based on discrete time. Think about how large this number needs to be in order to play notes of different durations in an abc piece. For example, if `ticksPerQuarterNote` had a value of 2, then an eighth note would be played for the duration of one tick, but you would not be able to schedule a sixteenth note for the correct duration.

play() After all of the notes have been scheduled, you can invoke `play` to tell the sequencer to begin playing the music.

Run the main method in `SequencePlayer`, which shows an example of using a sequencer to play up and down a C major scale. In this example, all of the notes in the scale have been hard-coded. In your abc player, you will be walking over your data structures that represent a musical piece and automatically schedule the notes at appropriate ticks.

We will come back to the sequencer in the next part of the lab.

abc Music Notation

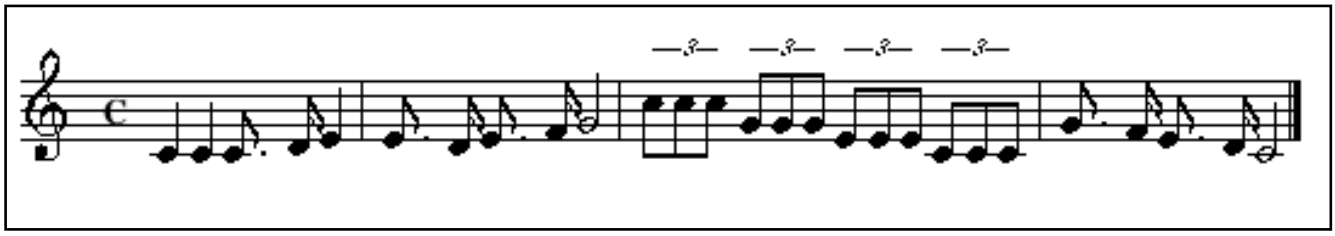
In this part of the lab, you will familiarize yourself with various language constructs by writing small, toy abc files and playing them using the Java MIDI sequencer.

This is a group exercise. Find your team members and work through this section together.

Task 1: Transcribe each of the following small pieces of music into an abc file. Name your files as **piece1.abc** and **piece2.abc**, respectively, and commit them under the directory `sample_abc` in your team's SVN repository.

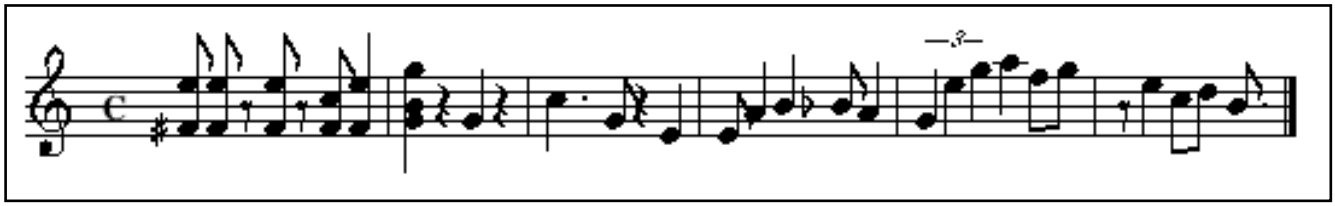
You may find the abc subset description useful.

Piece No.1: A simple, 4/4 meter piece with triplets. As a starter, the header and the first two bars are already provided. You should complete the rest of the piece by transcribing the last two bars.



```
X: 1
T:Piece No.1
M:C
L:1/4
Q: 140
K:C
C C C3/4 D/4 E | E3/4 D/4 E3/4 F/4 G2 |
```

Piece No.2: A more complex piece, with chords, accidentals, and rests. **Set its tempo to 200, with the default note length of 1/4.**



Task 2: Write JUnit tests that play these pieces using the sequencer, similar to the main method in the `SequencePlayer` class. Store them in a separate class called **`SequencePlayerTest`**.

> **Checkpoint.** Find a TA or another member of the course staff, and demonstrate your abc file creations by playing the two pieces on your computer.

Visitor Pattern

As discussed during the previous lecture, the Visitor pattern is a way to separate operations from data structures, such that new operations can be added without having to modify the structures. You may find the Visitor pattern useful for this project as you traverse over the musical structures and perform a variety of operations. In this part of the lab, you will practice using the Visitor pattern by applying it to simple, arithmetic expressions.

This is an individual exercise.

First, **check out** the project named **`traversal_java`** from your personal SVN repository.

Look at the abstract base class, `Expr` and its subclasses, `Const`, `Plus`, `Minus`, `Mult`, and `Div`, as well as the Visitor interface, `ExprVisitor`. The class `Const` is a wrapper class for the primitive Java type `double`.

Task 3: Write an `EvaluateVisitor` that walks over an expression and calculates its value in double. You may also want to modify the `Expr` classes. Create three JUnit test cases for the visitor and store them in `ExprVisitorTest.java`.

Task 4: Define an enumerated type called `Sign` that can take one of the three values, `POSITIVE`, `NEGATIVE`, and `UNKNOWN`. Write a `SignVisitor` that returns the sign of an expression **without** actually evaluating the expression. Note that your visitor should return

`POSITIVE` or `NEGATIVE` only if it can determine the sign of the overall expression purely from the arithmetic operators and the signs of the constants; otherwise, it should return `UNKNOWN`. You may also modify the `Expr` classes as necessary. Write three JUnit test cases and store them in `ExprVisitorTest` as well.

How would you have added the sign operation without using a visitor? What kind of changes would you have made to the `Expr` classes?

> ✔**Checkpoint.** Find a TA or another member of the course staff, and demonstrate your visitors.

Commit Your Solutions

This is the end of the lab. Be sure to commit all your code to your individual (`traversal_java`) **AND** group (`abcPlayer`) SVN repositories. **Since both you and your team members are sharing the same repository, be careful to check for conflicts when you commit, and don't break the build for your partners.**