6.005 Elements of Software Construction | Fall 2011
Problem Set 2: Midi Piano

The purpose of this problem set is to give you a chance to design with state machines, and to see one way of translating such a design into code.

The code in the later questions depends on the code in question 1 to work. Questions 2, 3, and 4 can be completed in any order. Question 5 depends on Question 4. Question 4 may take more time to complete than the previous questions.

Do not change the signatures or specifications of any methods, classes, or packages that we have provided you. Your code will be tested automatically, and will break our testing suite if you do so.

Overview

Throughout this problem set we will be implementing a simple midi keyboard. We will give it the capability to:

1. Play notes using a row of keys on your computer keyboard
2. Change among a number of instruments
3. Change octaves
4. Record sequences of notes and play them back.

We have provided you with code that wraps a midi device, some abstractions for dealing with musical notes, and some Java Applet code that listens for keypresses. Our applet code calls several methods of PianoMachine; you should keep those methods as the entry points for PianoMachine and not change their signatures or specs (as we will use these functions for automated testing.) Do not modify any code inside the 'midi' or 'music' packages. For the first four questions, do not modify any code in the PianoApplet class. Please note that we have provided a method, Midi.history(), which gives a text output from the piano and can be used for testing.

A slight annoyance: you'll need to click on the applet's panel with your mouse to give it keyboard focus before playing any notes.

Problem 1: Piano Keys

Our midi piano should allow us to play a set of pitches {C, C#, D, ... A#, B} using the keys {'1', '2', ... '-', '='} respectively. When one of these keys is pressed, a note should begin if it isn't already sounding; likewise, when such a key is released, a note should end if it is currently sounding. We provide method signatures for PianoMachine.beginNote and PianoMachine.endNote, which will be triggered by appropriate key presses and releases. Use those methods as entry points in your implementation.

a. [5 points] Write the specs for PianoMachine.beginNote and PianoMachine.endNote, and for any other helper methods you expect to need.

b. [5 points] Write test cases for these methods. You will find Midi.history() useful for this and other tests in this assignment.

c. [15 points] Add this functionality by implementing the beginNote and endNote methods, as well as any helper methods, and adding any necessary state to PianoMachine. To start you off, we've given a provisional implementation of beginNote and endNote which always plays middle 'C'.

Problem 2: Switching Instruments

The midi piano should be able to switch instruments. The 'I' key should switch our instrument mode to the next instrument in a list, or back to the start if we're at the end. We have mapped the 'I' key to PianoMachine.changeInstrument; use this method in your implementation.

a. [5 points] Write the spec for the changeInstrument method, and for any helper methods you expect to need.

b. [5 points] Write test cases for these methods.

c. [10 points] Add state to the PianoMachine class which reflects the instrument mode, fill in the changeInstrument method, and update any other code necessary so that your piano can switch instruments as specified. You may find the Instrument enum in the package 'midi' to

be useful.

Problem 3: Switching Octaves

Pressing the 'C' and 'V' keys should shift the notes that the keys play down and up, respectively, by one octave (12 semitones). We should be able to shift by two octaves, maximum, in either direction from the starting pitches. We have mapped the appropriate keys to PianoMachine.shiftUp and PianoMachine.shiftDown; use these methods in your solution.

a. [5 points] Write the spec for the shiftUp and shiftDown methods, and for any helper methods you expect to need.

b. [5 points] Write test cases for these methods.

c. [10 points] Add this functionality by adding the appropriate state to PianoMachine, implementing the shiftUp and shiftDown methods, and updating any other methods necessary.

Problem 4: Recording and Playback

The piano should have the ability to record and playback sequences of notes, preserving the rhythm they were played with. When you make a new recording it should replace the previous one. 'R' should toggle record mode on and off, and 'P' should trigger playback. As before, we've provided you with signatures for PianoMachine.toggleRecording and PianoMachine.playback as entry points; use these methods.

a. [5 points] Write the spec for the toggleRecording and playback methods, and for any helper methods you expect to need.

b. [5 points] Write test cases for these methods.

c. [15 points] Add this functionality, by making PianoMachine keep track of the necessary state, implementing the playback and toggleRecording methods, and adding any other necessary code. You might find the NoteEvent class useful.

Note: try triggering some new notes while the piano is in playback mode. You may find that the notes all sound together after playback finishes, as a single chord. Consider why this might be the case. For this problem, we will not deduct points for this behavior.

Problem 5: Keyboard Input During Playback

[10 points] If your application exhibits the (undesirable) behavior we describe at the end of Problem 4c, eliminate that behavior to complete this problem. That is, in your implementation, keyboard input that occurs during playback should have no effect, either immediately or later. You may edit piano.PianoApplet for this part of the assignment.

Hint: You may find the method KeyEvent.getWhen() to be useful.

Optional Extension

This extension is optional and will not affect your grade in any way. The only reason to try it is to quench your burning desire to program.

Add a "chord mode" to your keyboard, where each key you press will trigger several notes at once. A straightforward mapping would be to trigger a major triad for each note (e.g. pitches C, E, and G, if you press '1'.) You may also wish you make a different harmonization, at your musical discretion.

Before You're Done...

Double check that you didn't change the signatures of any of the code we gave you.

Make sure the code you implemented doesn't print anything to System.out. It's a helpful debugging feature, but writing output is a definite side effect of methods.

Make sure you don't have any outdated comments in the code you turn in. In particular, get rid of blocks of code that you may have commented out when doing the pset, and get rid of any TODO comments that are no longer TODOs.

Make sure your code compiles, and all the methods you've implemented pass all the tests that you added.

Does your code compile without warnings? In particular, you should have no unused variables, and no unneeded imports.

Make sure you check the last version of your code in SVN is the version you want to be graded.

Try to make sure that your code conforms to standard Java naming conventions. That is, class names should be StartingUpperCamelCase, and variables and methods should be startingLowerCamelCase.

6.005 Elements of Software Construction

Fall 2011