**Notes on JavaSound application**

JavaSound.java

Implements Runnable so it runs in a separate thread

Maintains a vector called demos which include an instance of Juke.java. There are 4 different demos in the applications (Juke, MidiSynth, CapturePlayback,

**Juke.java**

Implements:

Runnable - so it runs in a separate thread

LineListener – Java API interface: “Instances of classes that implement the LineListener interface can register to receive events when a line's status changes.”

MetaEventListener – Java API interface: “The MetaEventListener interface should be implemented by classes whose instances need to be notified when a [Sequencer](https://docs.oracle.com/javase/7/docs/api/javax/sound/midi/Sequencer.html) has processed a [MetaMessage](https://docs.oracle.com/javase/7/docs/api/javax/sound/midi/MetaMessage.html). To register a MetaEventListener object to receive such notifications, pass it as the argument to the [addMetaEventListener](https://docs.oracle.com/javase/7/docs/api/javax/sound/midi/Sequencer.html#addMetaEventListener(javax.sound.midi.MetaEventListener)) method of Sequencer.”

Sequencer – Java API interface: A hardware or software device that plays back a MIDI [sequence](https://docs.oracle.com/javase/7/docs/api/javax/sound/midi/Sequence.html) is known as a *sequencer*. A MIDI sequence contains lists of time-stamped MIDI data, such as might be read from a standard MIDI file. Most sequencers also provide functions for creating and editing sequences.

The Sequencer interface includes methods for the following basic MIDI sequencer operations:

* obtaining a sequence from MIDI file data
* starting and stopping playback
* moving to an arbitrary position in the sequence
* changing the tempo (speed) of playback
* synchronizing playback to an internal clock or to received MIDI messages
* controlling the timing of another device

In addition, the following operations are supported, either directly, or indirectly through objects that the Sequencer has access to:

* editing the data by adding or deleting individual MIDI events or entire tracks
* muting or soloing individual tracks in the sequence

notifying listener objects about any meta-events or control-change events encountered while playing back the sequence.

MetaMessage - Java API interface: A MetaMessage is a [MidiMessage](https://docs.oracle.com/javase/7/docs/api/javax/sound/midi/MidiMessage.html) that is not meaningful to synthesizers, but that can be stored in a MIDI file and interpreted by a sequencer program.

MidiMessage is the base class for MIDI messages. They include not only the standard MIDI messages that a synthesizer can respond to, but also "meta-events" that can be used by sequencer programs. There are meta-events for such information as lyrics, copyrights, tempo indications, time and key signatures, markers, etc. For more information, see the Standard MIDI Files 1.0 specification, which is part of the Complete MIDI 1.0 Detailed Specification published by the MIDI Manufacturer's Association ([http://www.midi.org](http://www.midi.org/)).

ControlContext – Not a Java API class or interface. Defined in the JavaSound application as: “The interface for the JavaSound tabs to open and close audio resources.” It just contains the two functions open and close with no arguments and void returns.

Juke gets implemented and passed the audio directory

**Note: A lot of Juke is just eye-candy that has no relationship to actually playing the audio files.**

JukeControls – class defined inside of Juke.java. It is a JPanel with the start, stop, and pause buttons which acts as its own action listener for those button. So it implements the actionPerformed function.

Start/Stop button

When text is “Start” calls the JukeControls.start() and changes text to “Stop”.

When text is “Stop” calls the JukeControls.stop() and changes text to “Start”.

Pause/Resume button

When text is “Pause” performs actions below and changes text to “Resume”.

If the current audio being played is a clip calls clip.stop()

If the current audio is an instance of Sequence or BufferedInputStream calls sequencer.stop(). The sequencer is an instance of Sequence. See description below.

After doing one of the above calls playbackMonitor.stop().

When text is “Resume” performs actions below and changes text to “Pause”.

If the current audio being played is a clip calls clip.start()

If the current audio is an instance of Sequence or BufferedInputStream calls sequencer.start().

After doing one of the above calls playbackMonitor.start().

There are also handlers for the “>>” and “<<” buttons which are assumed to be fast forward and fast reverse.

Sequence – Java class: A Sequence is a data structure containing musical information (often an entire song or composition) that can be played back by a [Sequencer](https://docs.oracle.com/javase/7/docs/api/javax/sound/midi/Sequencer.html) object. Specifically, the Sequence contains timing information and one or more tracks. Each [track](https://docs.oracle.com/javase/7/docs/api/javax/sound/midi/Track.html) consists of a series of MIDI events (such as note-ons, note-offs, program changes, and meta-events). The sequence's timing information specifies the type of unit that is used to time-stamp the events in the sequence.

A Sequence can be created from a MIDI file by reading the file into an input stream and invoking one of the getSequence methods of [MidiSystem](https://docs.oracle.com/javase/7/docs/api/javax/sound/midi/MidiSystem.html). A sequence can also be built from scratch by adding new Tracks to an empty Sequence, and adding [MidiEvent](https://docs.oracle.com/javase/7/docs/api/javax/sound/midi/MidiEvent.html) objects to these Tracks.

Is it possible to create a Sequence from an audio file then put in it MidiEvents to trigger changing slides in the slide show?

**Functions in Juke**

public boolean loadSound(Object object)

If object is an instance of URL takes care of loading from a web page. Don’t need this for slide show.

Else if object is an instance of File

Get the name of the file by calling file.getName() (name only not path)

In a try block calls currentSound = AudioSystem.*getAudioInputStream*((File) object); where currentSound is just defined as Object currentSound.

If the try fails then in the catch is another try block that calls currentSound = **new** BufferedInputStream(is, 1024);

If that try fails the function prints a stack trace and returns false.

If still here call loading.interrupt(); **Not sure what this does.**

If sequencer is null here then it means the open() function was not called to initialize it so exit the function.

If currentSound instance of AudioInputStream then it means we are playing a clip so set up for that.

Else if currentSound instance of Sequence or BufferedInputStream

In a try block…

call sequencer.open() – sequencer is an instance of Sequencer which is an interface. This is set to the Sequencer object returned by a call to MidiSystem.getSequencer in the open function of Juke.

Based on whether the currentSound is a Sequencr or a BufferedInputStream call sequencer.setSequence with the appropriate object type.

If that didn’t work then two catches exit the function

If everything worked then initialize the seekSlider to 0 and enable all the slides to control advancing or controlling the pan (???) and gain.

playSound

Call playbackMonitor.start()

Call setGain, setPan, and set several Boolean flags (midiEOM, audioEOM, and bump) to false.

If currentSound is a Sequene or BufferedInputStream and the thread not null

Call sequencer.start()

while(!midiEOM && thread != null && !bump)

In a try block sleep the thread for 99

**It would appear this is where the sound playing is started and this is the thread pause loop to wait for the playing to complete.**

When the while loop exits then call sequencer.stop and sequencer.close.

**Check these out…**

midiEOM gets set to true in the function Juke.meta(MetaMessage message)

bump gets set to true if the “>>” or “<<” buttons are clicked.

**Class PlaybackMonitor**

Class defined in Juke starting at line 806