JAMMING WITH JAVA™ TECHNOLOGY: MAKING MUSIC WITH JFUGUE AND JFRETS

David Koelle, Author of JFugue
Senior Software Engineer, Charles River Analytics Inc

Matt Warman, Author of JFrets
Senior Software Engineer, STAR BASE Consulting Inc

TS-5263

Java

Sun microsystems

Learn how to create music from within your own Java™ application

Recapture the joy of programming

GOAL

Agenda

- Introduction to JFugue
- Making Music with JFugue
- JFugue's Advanced Features
- JFugue Under the Hood
- Applications of JFugue
- JFugue and JFrets

Introduction to JFugue

- JFugue is an open-source API for programming music in Java code
- Prevents you from dealing with MIDI messages!
  - But generates MIDI behind the scenes
- Allows you to specify music naturally
  - `player.play("C D E F G A B");`
- Provides classes that make music exploration fun and easy
  - Microtonal music
  - Rhythms
  - Interacting with external devices
  - Many other easy-to-use features
- Enables interaction with other music tools and formats
  - Read or write musical data from MIDI, MusicXML, etc.
  - Extensible architecture

How hard is music programming?
Music without JFugue

```
// Play a Middle-C
Sequencer sequencer = MidiSystem.getSequencer();
Sequence sequence = sequencer.getSequence();
Track track = sequence.createTrack();
ShortMessage onMessage = new ShortMessage();
onMessage.setMessage(ShortMessage.NOTE_ON, 0, 60, 128);
MidiEvent noteOnEvent = new MidiEvent(onMessage, 0);
track.add(noteOnEvent);
ShortMessage offMessage = new ShortMessage();
offMessage.setMessage(ShortMessage.NOTE_OFF, 0, 60, 128);
MidiEvent noteOffEvent = new MidiEvent(offMessage, 200);
track.add(noteOffEvent);
sequencer.start();
try {
    Thread.sleep(track.ticks());
} catch (InterruptedException e) {
    Thread.currentThread().interrupt();
}
```

How hard is music programming?
Music without JFugue – another way

```
// Play a Middle-C
try {
    Synthesizer synthesizer = MidiSystem.getSynthesizer();
    synthesizer.open();
    MidiChannel[] channels = synthesizer.getChannels();
    channels[0].noteOn(60, 128);
    try {
        Thread.sleep(200);
    } catch (InterruptedException e)
    {
        // handle exception
    }
    channels[0].noteOff(60);
    synthesizer.close();
} catch (MidiUnavailableException e) {
    // handle exception
}
```

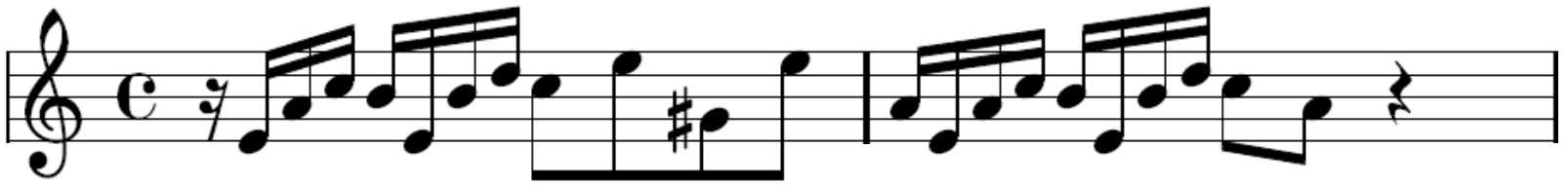
Make it easy!
Music with JFugue

```
// Play a Middle-C
Player player = new Player();
player.play("C");
```

Wow, only 2 lines of code?

- The Magic of JFugue: Music is specified using JFugue's "MusicString"
  - `player.play("C")` is a simple case
  - J. S. Bach's *Inventio 13*

```
player.play("E5s A5s C6s B5s E5s B5s D6s C6i E6i G#5i
E6i | A5s E5s A5s C6s B5s E5s B5s D6s C6i A5i Ri");
```



- JFugue creates all of the MIDI messages behind the scenes

But… it creates MIDI… that's so *1980*

- Problem: You heard the notes, but it doesn't sound like *Music*
- Bias: MIDI is dead. MP3 rules!
  - But MIDI specifies musical messages; MP3 is a compression format
- Cause: The MIDI synthesizer doesn't have a good soundbank
  - People tend to associate MIDI with bad-sounding music
  - But MIDI is just a specification for musical messages
  - What you're hearing is messages turned into sound using synthesizer
  - Soundbanks are replaceable, thanks to the Audio Synthesis Engine Project http://openjdk.java.net/projects/audio-engine/
- Solution: Use a synthesizer that can load better soundbanks!
  - There are a lot of soundbanks in the world that sound *fantastic*!
- Use Gervill, an audio synthesis engine, to load new soundbanks https://gervill.dev.java.net/

Code for adding Gervill to JFugue

```
// There is no code!
//
// Just add gervill.jar to your classpath,
// and MidiSystem.getSynthesizer() will return
// an instance of a Gervill Synthesizer.
```

Code for loading a better soundbank

```
// Really, the hardest part is finding
// a soundbank you like.
Soundbank soundbank =
    MidiSystem.getSoundbank(new File("filename"));

// Load instruments from the soundbank
// into the synthesizer
Synthesizer synth = MidiSystem.getSynthesizer();
synth.loadAllInstruments(soundbank);

// Create a JFugue Player object that is attached
// to the synthesizer with the new instrument
Player player = new Player(synth);

// Now play your music with better results!
player.play(your music here);
```

Using a better soundbank

- Find new soundbanks… Some are free, some cost money
- My favorite: **SONiVOX's 250 Meg GM Wavetable**
  http://www.sonivoxrocks.com/

SONiVOX
sound that rocks.

- Sample of MIDI music rendered using SONiVOX soundbanks:
- Revisiting *"Inventio 13"*

Using standard soundbank

Using **SONiVOX 250 Meg GM Wavetable** soundbank

Agenda

- Introduction to JFugue
- Making Music with JFugue
- JFugue's Advanced Features
- JFugue Under the Hood
- Applications of JFugue
- JFugue and JFrets

Making Music with JFugue
Two building blocks

- Programming music in JFugue comes down to two key ideas:

- **MusicString** – a String that contains notation for specifying music
  - Example: `"C5q E5q Cmajq"`

- **Pattern** – a class that allows MusicStrings to be built, altered, recombined, etc.
  - Example:

```
Pattern pattern = new Pattern("C5q E5q Cmajq");
pattern.add("D5s");
```

  - Most features in JFugue API return instances of Pattern

Specifying Music in JFugue

The JFugue MusicString

- Three rules of thumb when using JFugue:
  - If it can be specified in MIDI, JFugue can create it
  - If it requires memorization, JFugue has predefined constants
  - If it takes work to make it right, JFugue simplifies it

Specifying Music in JFugue

Anatomy of a MusicString

```
player.play("T[Adagio] V0 I[Piano] C5q F#5q
CmajQ V1 I[Flute] C3q+E3q E3q+G3q Ri
C2majI");
```

Specifying Music in JFugue

Anatomy of a MusicString

```
player.play("T[Adagio] V0 I[Piano] C5q F#5q
CmajQ V1 I[Flute] C3q+E3q E3q+G3q Ri
C2majI");
```

- Tempo
  - Indicates speed of music
  - Letter T followed by Beats Per Minute (as of JFugue 4.0)
  - Pre-defined constants, like ‘Adagio’ or ‘Largo’, also provided

Specifying Music in JFugue

Anatomy of a MusicString

```
player.play("T[Adagio] V0 I[Piano] C5q F#5q
CmajQ V1 I[Flute] C3q+E3q E3q+G3q Ri
C2majI");
```

- Voice
  - Specifies MIDI channel for subsequent notes and other musical events
  - Letter V followed by one of the 16 MIDI channels (0-15)
  - The 10th voice (V9) is special – that's MIDI's percussion track

Specifying Music in JFugue

Anatomy of a MusicString

```
player.play("T[Adagio] V0 I[Piano] C5q F#5q
CmajQ V1 I[Flute] C3q+E3q E3q+G3q Ri
C2majI");
```

- Instrument
  - Selects which instrument to use for playing music
  - Letter I followed by a number from 0-127, representing the 128 MIDI instruments
  - Pre-defined constants, like “Piano” and “Flute”, are provided for each of the MIDI instruments

Specifying Music in JFugue

Anatomy of a MusicString

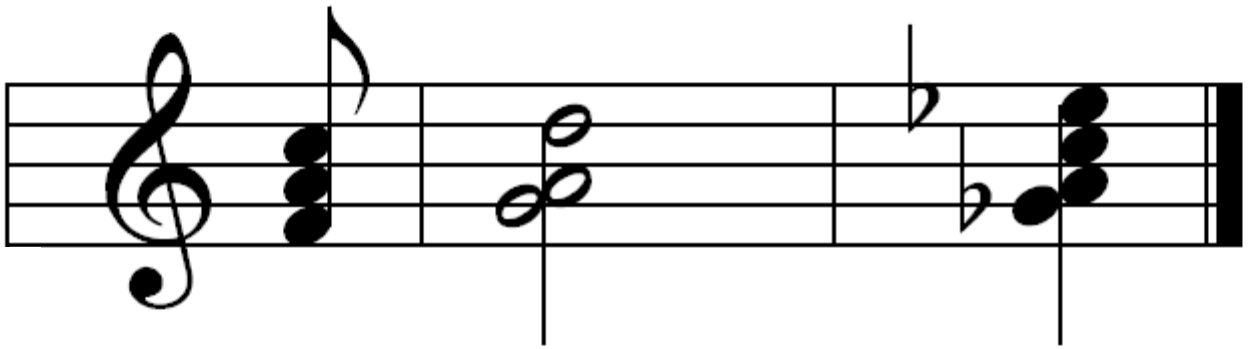
```
player.play("T[Adagio] V0 I[Piano] C5q F#5q
CmajQ V1 I[Flute] C3q+E3q E3q+G3q Ri
C2majI");
```

- Notes, Rests, and Chords
  - Note letter (C, D, E, F, G, A, or B), accidental (#, b, natural), and octave
    - Rest is specified with an R
  - Duration: w, h, q, i, s, t, x, o (whole, half, etc. down to 128th)
    - Dotted notes, tuplets, ties, and combined durations are all supported
  - Note and duration can each be specified numerically: [60], C5/0.5
  - Notes in harmony indicated with +: C3q+E3q
- Chords
  - Root note plus chord identifier (maj, min, aug, etc), then duration
  - Chord inversions can be specified with ^

Specifying Music in JFugue

Anatomy of a MusicString

- JFugue supports 30 types of chord (maj, min, aug, etc)
- Chord inversions are specified with ^



FmajQ Gsus2H Cdim7^^Q

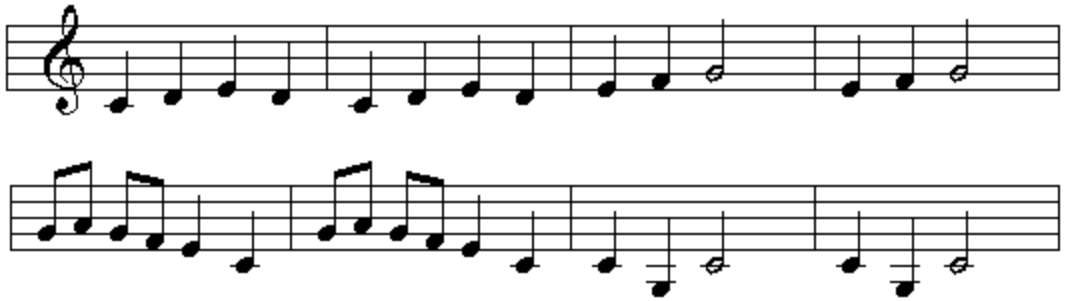
Specifying Music in JFugue

Additional MusicString commands

- Key Signature
  - Letter K followed by a key. Examples: KAbmin, KFmaj
  - All notes in the composition will be automatically played in the key
    - Example: a B note in an F-major key will be converted to B-flat.
- Timing Information
  - @ followed by a time in milliseconds to play next token
  - Seen especially when parsing MIDI files
- Grab bag of MIDI events
  - Pitch Wheel – useful for getting microtones out of MIDI!
  - Channel Pressure
  - Polyphonic Pressure
  - Controller Events – JFugue combines low- and high-bytes into one token

JFugue's Patterns
Music is Poetic

- Music frequently has repeated phrases
- "Frere Jacques"



- Patterns allow common bits of music to be re-used:

```
Pattern pattern1 = new Pattern("C5q D5q E5q C5q");
Pattern song = new Pattern();
song.add(pattern1, 2); // Adds 'pattern1' to 'song' twice
```

JFugue's Patterns
Music is Poetic

- Patterns are more than just MusicStrings…
- …Patterns can be altered in interesting ways
  - Reverse, invert, change durations, change pitches, swap instruments, etc.
- J. S. Bach's "Crab Canon" from "The Musical Offering"
  - Two players, playing simultaneously
  - The second player plays a mirror image of the first player's notes
  - See Douglas R. Hofstadter's "Gödel, Escher, Bach"
- CrabCanon in JFugue: (MusicString available at http://www.jfugue.org)

```
Pattern voice1 = new Pattern(notes for one voice);
Pattern voice2 = new
     RevesePatternTransformer.transform(voice1);
```

Agenda

- Introduction to JFugue
- Making Music with JFugue
- JFugue's Advanced Features
- JFugue Under the Hood
- Applications of JFugue
- JFugue and JFrets

Microtonal Music

"The notes between the cracks" – Charles Ives

- Occurrences of microtones:
  - Eastern music (Indian, Turkish)
  - Gamelan (Javanese, Balinese)
  - Modernist compositions (Charles Ives, Philip Glass)
  - Musical effects – portamento, slide trombone

- MIDI is capable of playing microtones, but it's hard to do
  - Requires combination of Note and Pitch Wheel events, and lots of math
- JFugue… wait for it… makes it easy!

- Three steps:
  - 1. Assign frequencies to microtonal notes
  - 2. Define your music
  - 3. Generate a Pattern and play it

Microtonal Music

```
public static void main(String[] args) {
 MicrotoneNotation microtone = new MicrotoneNotation();
 microtone.put("A440", 440.00); microtone.put("z3",   704.00);
 microtone.put("z1",   528.00); microtone.put("z4",   792.00);
 microtone.put("z2",   616.00); microtone.put("A880", 880.00);

 String micro1 = "<A440>s Rt <A440>s Rt <z1>s Rt <z1>s Rt";
 String micro2 = "<A440>q.";
 String micro3 = "<A880>t <z3>t <z4>t <z2>t <z3>t <z2>t”;

 Pattern pattern = new Pattern();
 pattern.add("V0 I[SKAKUHACHI]");
 pattern.add(microtone.getPattern(micro1), 3);
 pattern.add(microtone.getPattern(micro2));
 pattern.add("V1 I[VOICE_OOHS]");
 pattern.add(microtone.getPattern(micro3), 8);

 new Player().play(pattern);
}
```

Microtonal Music

```
public static void main(String[] args) {
 MicrotoneNotation microtone = new MicrotoneNotation();
 microtone.put("A440", 440.00); microtone.put("z3",   704.00);
 microtone.put("z1",   528.00); microtone.put("z4",   792.00);
 microtone.put("z2",   616.00); microtone.put("A880", 880.00);

 String micro1 = "<A440>s Rt <A440>s Rt <z1>s Rt <z1>s Rt";
 String micro2 = "<A440>q.";
 String micro3 = "<A880>t <z3>t <z4>t <z2>t <z3>t <z2>t”;

 Pattern pattern = new Pattern();
 pattern.add("V0 I[SKAKUHACHI]");
 pattern.add(microtone.getPattern(micro1), 3);
 pattern.add(microtone.getPattern(micro2));
 pattern.add("V1 I[VOICE_OOHS]");
 pattern.add(microtone.getPattern(micro3), 8);

 new Player().play(pattern);
}
```

Microtonal Music

```
public static void main(String[] args) {
 MicrotoneNotation microtone = new MicrotoneNotation();
 microtone.put("A440", 440.00); microtone.put("z3",   704.00);
 microtone.put("z1",   528.00); microtone.put("z4",   792.00);
 microtone.put("z2",   616.00); microtone.put("A880", 880.00);

 String micro1 = "<A440>s Rt <A440>s Rt <z1>s Rt <z1>s Rt";
 String micro2 = "<A440>q.";
 String micro3 = "<A880>t <z3>t <z4>t <z2>t <z3>t <z2>t”;

 Pattern pattern = new Pattern();
 pattern.add("V0 I[SKAKUHACHI]");
 pattern.add(microtone.getPattern(micro1), 3);
 pattern.add(microtone.getPattern(micro2));
 pattern.add("V1 I[VOICE_OOHS]");
 pattern.add(microtone.getPattern(micro3), 8);

 new Player().play(pattern);
}
```

Intervals

- Specify music as intervals instead of actual notes
  - Use the difference between the notes, instead of the notes themselves
  - This is different than Key Signature
  - Intervals can be followed by chords, durations, etc.
- Example:

```
public static void main(String[] args) {
    IntervalNotation riff =
        new IntervalNotation("<1>q <5>q <8>q <1>majH");

    Player player = new Player();
    player.play(riff.getPatternForRootNote("C5"));
    player.play(riff.getPatternForRootBote("Ab6"));
}
```

Rhythms

- Define beats in a natural, intuitive manner
  - Use your computer's keyboard like a drum machine
- Example:

O..oO...O..oOO..

- Three steps:
  - 1. Bang out your beat
  - 2. Assign MusicStrings to keys
  - 3. Generate a Pattern and play it

Rhythms

```
// This is a complete program for a 16-Beat Rock Rhythm
public static void main(String[] args) {
    Rhythm rhythm = new Rhythm();
    rhythm.setLayer(1, "O..oO...O..oOO..");
    rhythm.setLayer(2, "..*...*...*...*.");
    rhythm.setLayer(3, "^^^^^^^^^^^^^^^^");
    rhythm.setLayer(4, "................!");

    rhythm.addSubstitution('O', "[BASS_DRUM]i");
    rhythm.addSubstitution('o', "Rs [BASS_DRUM]s");
    rhythm.addSubstitution('*', "[ACOUSTIC_SNARE]i");
    rhythm.addSubstitution('^', "[PEDAL_HI_HAT]s Rs”);
    rhythm.addSubstitution('!', "[CRASH_CYMBAL_1]s Rs");
    rhythm.addSubstitution('.', "Ri");

    Pattern pattern = rhythm.getPattern();
    pattern.repeat(4);
    Player player = new Player();
    player.play(pattern);
}
```

Rhythms

```
// This is a complete program for a 16-Beat Rock Rhythm
public static void main(String[] args) {
    Rhythm rhythm = new Rhythm();
    rhythm.setLayer(1, "O..oO...O..oOO..");
    rhythm.setLayer(2, "..*...*...*...*.");
    rhythm.setLayer(3, "^^^^^^^^^^^^^^^^");
    rhythm.setLayer(4, "...............!");

    rhythm.addSubstitution('O', "[BASS_DRUM]i");
    rhythm.addSubstitution('o', "Rs [BASS_DRUM]s");
    rhythm.addSubstitution('*', "[ACOUSTIC_SNARE]i");
    rhythm.addSubstitution('^', "[PEDAL_HI_HAT]s Rs”);
    rhythm.addSubstitution('!', "[CRASH_CYMBAL_1]s Rs");
    rhythm.addSubstitution('.', "Ri");

    Pattern pattern = rhythm.getPattern();
    pattern.repeat(4);
    Player player = new Player();
    player.play(pattern);
}
```

Rhythms

```
// This is a complete program for a 16-Beat Rock Rhythm
public static void main(String[] args) {
    Rhythm rhythm = new Rhythm();
    rhythm.setLayer(1, "O..oO...O..oOO..");
    rhythm.setLayer(2, "..*...*...*...*.");
    rhythm.setLayer(3, "^^^^^^^^^^^^^^^^");
    rhythm.setLayer(4, "...............!");

    rhythm.addSubstitution('O', "[BASS_DRUM]i");
    rhythm.addSubstitution('o', "Rs [BASS_DRUM]s");
    rhythm.addSubstitution('*', "[ACOUSTIC_SNARE]i");
    rhythm.addSubstitution('^', "[PEDAL_HI_HAT]s Rs”);
    rhythm.addSubstitution('!', "[CRASH_CYMBAL_1]s Rs");
    rhythm.addSubstitution('.', "Ri");

    Pattern pattern = rhythm.getPattern();
    pattern.repeat(4);
    Player player = new Player();
    player.play(pattern);
}
```

Rhythms and Intervals

- You can use Intervals in your Rhythms to create beats with riffs

```
rhythm.setLayer(1, "o...o...o...o...o...o...o...o...");
rhythm.setLayer(2, "..*...*...*...*...*...*...*...*.");
rhythm.setLayer(3, "...%...%...%...%...%...%...%...%");
rhythm.setVoice(1, "jjnnjjmlnnllnnlkjjnnjjmlkkklnnnk");

rhythm.addSubstitution('j', "<1>s Rs");
rhythm.addSubstitution('k', "<6>s Rs");
rhythm.addSubstitution('l', "<8>s Rs");
// etc.
```

- Now get the music for given a specific root note


```
Pattern pattern = rhythm.getPatternWithInterval("Bb4");
Pattern pattern = rhythm.getPatternWithInterval("A5");
```

Sharing Music with MIDI Devices
Talk to Your Musical Keyboard

- Sending music to an external device is very easy:

```
DeviceThatWillReceiveMidi device =
    new DeviceThatWillReceiveMidi(MidiDevice.Info);
sequence = player.getSequence(pattern);
device.sendSequence(sequence);
// Also: sequence = MidiSystem.getSequence(File);
```

- Reading music from an external device is very easy:

```
DeviceThatWillTransmitMidi device =
    new DeviceThatWillTransmitMidi(MidiDevice.Info);
device.listenForMillis(5000);
Pattern pattern = device.getPatternFromListening();
```

- Each of these handles many of lines of MIDI code

Loading and Saving Patterns and MIDI

- Patterns can be loaded and saved:
  - `pattern.savePattern(File)`
  - `Pattern pattern = Pattern.loadPattern(File)`
- Music can be saved as MIDI
  - `player.saveMidi(Pattern, File)`
- Music can be loaded from MIDI and converted into a Pattern!
  - `Pattern pattern = Pattern.loadMidi(File)`
- JFugue can also be load and save MusicXML
  - Other formats on the horizon!

Agenda

- Introduction to JFugue
- Making Music with JFugue
- JFugue's Advanced Features
- JFugue Under the Hood
- Applications of JFugue
- JFugue and JFrets

JFugue Under the Hood
Parsers & ParserListeners

- JFugue parses musical notation, and generates musical events
- Originally, JFugue parsed MusicStrings and generated MIDI
- Then, JFugue could parse MIDI and generate MusicStrings
  - Get – and play with – JFugue notation for your favorite MIDI files!
- Most recently, JFugue can parse and render MusicXML files
  - Thanks to contributions from the community!
- All of these are interchangeable!

JFugue Under the Hood
Parsers & ParserListeners

- A Parser knows how to convert data into musical events
- A ParserListener knows how to handle/render musical events

- Code to connect Parsers and ParserListeners:

```
YourParser parser = new YourParser();
YourRenderer renderer = new YourRenderer();
parser.addParserListener(renderer);
parser.parse(whatever object the parser can parse);
```

JFugue Under the Hood
Fun with Parsers & ParserListeners

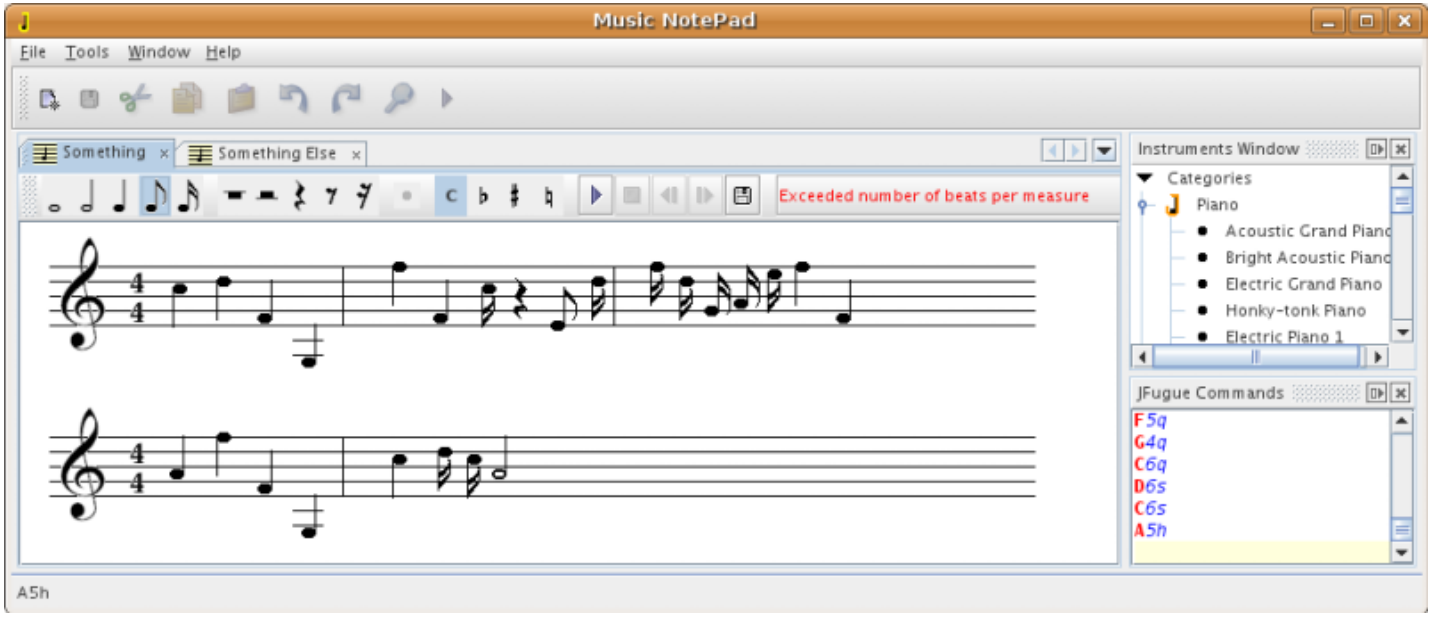
- Parsers and ParserListeners aren't limited to musical formats!
- Potential types of Parsers or ParserListeners:
  (none of these exist today)
  - SheetnoteRenderer – convert music to a graphical sheet of music
  - VisualizationRenderer – create fancy graphics based on the music
  - SpamParser – take junk email and convert it to music

Agenda

- Introduction to JFugue
- Making Music with JFugue
- JFugue's Advanced Features
- JFugue Under the Hood
- Applications of JFugue
- JFugue and JFrets

JFugue Music NotePad

- A simple, standalone application for creating music
  - Uses JFugue API; generates JFugue MusicStrings
- User interface is based on the NetBeans platform
- Entirely separate open-source project from JFugue
  - Started by Geertjan Weilenga – http://blogs.sun.com/geertjan/
  - Website: https://nbjfuguesupport.dev.java.net/



JFugue + Ant
Hear the status of your build

- From Geertjan Weilenga's blog
  http://blogs.sun.com/geertjan/entry/ode_to_build_scripts

- AntLogger class can listen to AntEvents
- Play a JFugue pattern when the build finishes
  - Happy tune for a successful build
  - Dark tune for a failed build

JFugue + Ant

```
import org.apache.tools.ant.module.spi.AntLogger;

public class BuildLogger extends AntLogger {

  public void buildFinished(AntEvent event) {
    Player player = new Player();
    Throwable t = event.getException();
    if (t != null) {
      // There has been an exception
      event.getSession().println(t.toString(), true, null);
      player.play("I[String_Ensemble_1] B3q Bb3q. G3i F3h");
    } else {
      // Build was successful
      player.play("I[French_Horn] As E6h As E6i Rt As E6h");
    }
  }

  public boolean interestedInSession(AntSession session) { return true; }
  public boolean interestedInAllScripts(AntSession session) { return true; }
}
```

When Programs Create Music

Use JFugue to Make Music Programmatically

- What might you create if you could programmatically define music?
  - *There's* something you can't do with a graphical music editor!

JFugue Drum Circle
We Got the Beat

- Start with JFugue's Rhythm class, with layers set to empty

t_0: `...............` `...............` `...............`

- Gradually and randomly add and remove 'strikes' from layers

t_1: `.a.............` `........b..b...` `..c.....cc......`
t_2: `.a...a....a.....` `..b.....b..b...` `...c....cc......`
t_3: `.a...aa...a...a.` `..bb.....b.....` `...c....c.......`
t_4: `.....a....a...a.` `..bb.....b....b.` `...cc...c....c..`

- Assign tones to strikes

```
rhythm.addSubstitution("a", "[HAND_CLAP]i");
rhythm.addSubstitution("b", "[BASS_DRUM]i");
rhythm.addSubstitution("c", "[LOW_BONGO]i");
```

- Play the rhythm!

```
player.play(rhythm.getPattern());
```

“The Sound of Shopping”
Turning Barcodes into Music

- Interactive exhibit turns barcodes, date and time of purchase into sequence of numbers
- A human composer specifies how numbers are made into music

Date start = 2008-02-08 14:42:10
Date end = 2008-02-08 14:42:47
ID = AA020
Barcodes = [54353]
K-Code =
770837452031941401436563400765351575415656506754345837825622920046164328 7
Ruleset = RuleSet4
Instrumentset = InstrumentSet4_3
JFugue = T113V15 [76]qia87 [73]sa91 Rs [69]qia79 [73]sa77 [78]qa75 [73]hqa79 [68]sa67 [71]sa80 Ri [73]sa80 [75]sa80 Rs [76]hqa83 [69]ha75 Rq [75]ha91 [78]ha75 [76]hqa95 [80]hqa75 [73]qia79 Rs . . .

- http://www.soundofshopping.com

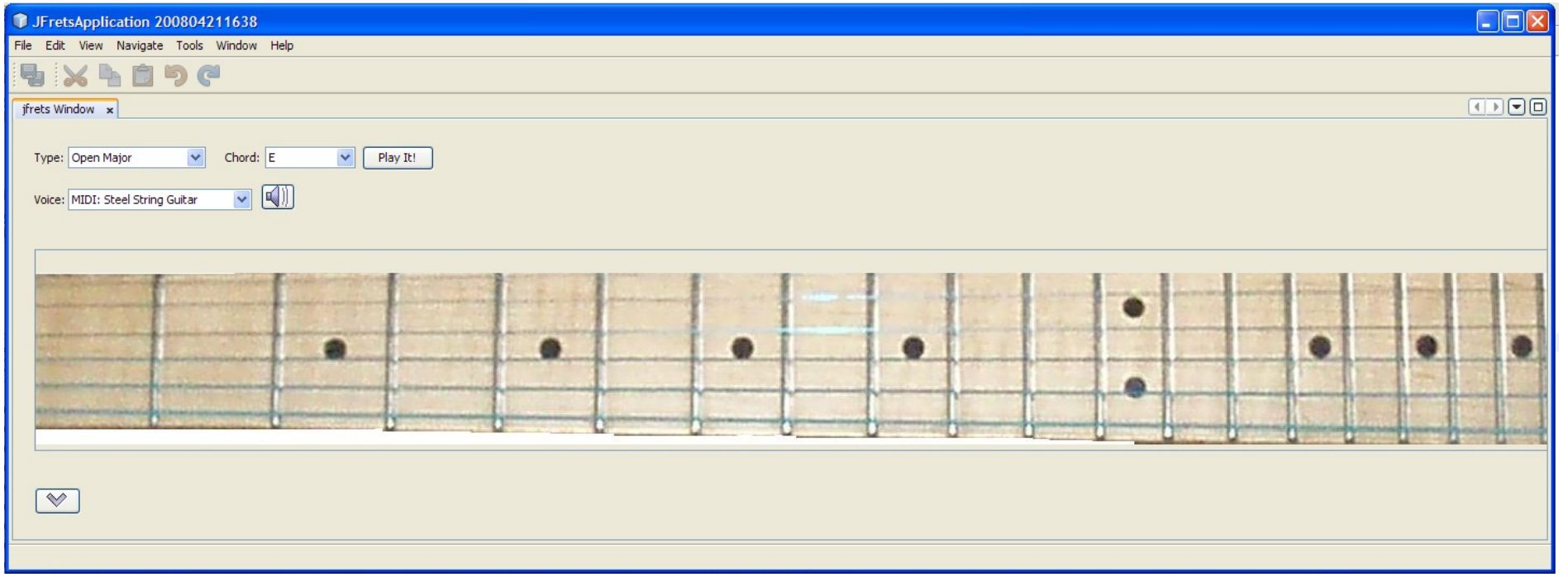
Agenda

- Introduction to JFugue
- Making Music with JFugue
- JFugue's Advanced Features
- Applications of JFugue
- JFugue Under the Hood
- JFugue and JFrets

What Is JFrets?

- Teaches guitar in an interactive desktop tool
- Displays notes, chords, and scales
- Plays sounds to aid in learning process
- Provides tutorials and exercises
- Ability to create and save guitar tablature
- Guitar tablature playback

What is JFrets?



How is JFugue used in JFrets?

- User selects MIDI voice
- User selects a note/chord
- User can set the Beats Per Minute
- The selection is parsed into a JFugue format and played

JFugue Code

```
Player player = new Player();
String patternString =
  getMidiType((String)this.voiceBox.getSelectedItem()) +
  noteName;
patternString = "T" + bpm + " " + patternString;
Pattern pattern = new Pattern(patternString);
player.play(pattern);
```

Audio Streaming Code

```
try
{
    File audio = new File(getClass().getResource("/org/jfrets/sounds/" + file).toURI());
    float sample = 128000;
    AudioInputStream ais = AudioSystem.getAudioInputStream(audio);
    AudioFormat af = ais.getFormat();
    AudioFormat target = new AudioFormat(AudioFormat.Encoding.PCM_SIGNED, af.getSampleRate(), 16,
    af.getChannels(), af.getChannels() * 2, af.getSampleRate(), false);
    AudioInputStream decode = AudioSystem.getAudioInputStream(target, ais);
    DataLine.Info info = new DataLine.Info(SourceDataLine.class, target);
    SourceDataLine line = (SourceDataLine) AudioSystem.getLine(info);
    line.open(target);
    if (line != null)
    {
         byte[] data = new byte[4096];
         line.start();
         int bytesRead;
         while ((bytesRead = decode.read(data, 0, data.length)) != -1)
         {
            line.write(data, 0, bytesRead);
         }
         line.drain();
         line.stop();
         line.close();
         decode.close();
    }
} catch (IOException e) {
    /* handle this exception */
} catch (LineUnavailableException e) {
    /* handle this exception */
} catch (URISyntaxException e) {
    /* handle this exception */
} catch (UnsupportedAudioFileException e) {
    /* handle this exception */
}
```

Don't worry, you're not supposed to be able to read this.

And you certainly shouldn't have to program it.

JFrets Capabilities

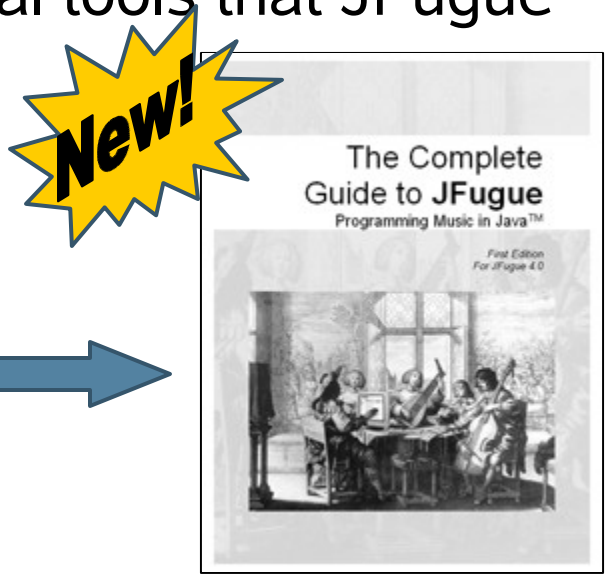
- Saves, displays, and plays guitar tablature
- Creates songs in tab or note format
- Provides a Metronome
- Provides a Scale Player
- Contains a guitar tuner with various tunings
- Prints tabs or songs

JFrets Demo

Summary

- JFugue makes music programming easy and fun
  - Create exciting new musical things!
  - Get your kids interested in programming!
  - Impress your co-workers!
  - Rekindle your joy of programming!
- JFrets exemplifies the kind of musical tools that JFugue can help make possible
- Java API Rocks!

- Project websites
  - JFugue – http://www.jfugue.org
  - JFrets - https://jfrets.dev.java.net



THANK YOU

David Koelle, Senior Software Engineer, Charles River Analytics Inc
Matt Warman, Senior Software Engineer, STAR BASE Consulting Inc

TS-5263