Structuring Music

CS1316: Representing Structure and Behavior

Story

- Using JMusic
  - With multiple Parts and Phrases
  - Creating music objects for exploring composition
    - Version 1: Using an array for Notes, then scooping them up into Phrases.
    - Version 2: Using a linked list of song elements.
    - Version 3: General song elements and song phrases
      - Computing phrases
      - Repeating and weaving
    - Version 4: Creating a tree of song parts, each with its own instrument.

JMusic: Java Music library

- JMusic knows about WAV files and many other formats, too (e.g., QuickTime)
- We’ll use it for manipulating MIDI
  - Musical Instrument Digital Interface, an industry-standard interface used on electronic musical keyboards and PCs for computer control of musical instruments and devices.
  - MIDI is about recording music, not sound.

Creating Notes

- Welcome to DrJava.
  - import jm.music.data.*
  - import jm.JMC;
  - import jm.util.*;
  - Note n = new Note(JMC.C4, JMC.QUARTER_NOTE);
  - n
  - Note n2 = new Note(64, 2.0);
  - n2

JMC=JMusic Constants

- Makes code easier to read from a music perspective

Creating Phrases

- Phrase phr = new Phrase();
- phr.addNote(n);
- phr.addNote(n2);
- double [] notes1 = {67, 0.25, 64, 0.5, 60, 1.0}
- phr.addNoteList(notes1)
- double [] notes2 = {JMC.G4, JMC.QN, JMC.E4, JMC.EN, JMC.C4, JMC.WN}
- phr.addNoteList(notes2)

Viewing Phrases

- View.notate(phr)
From Viewer: Manipulate and MIDI

- Can save or open MIDI files
- Can change key or time signature.
- Other tools allow changing other characteristics, like tempo.

Different ways of creating Phrases

> Phrase phr2 = new Phrase("Phrase 2",4.0,JMC.FLUTE);
> phr2.addNoteList(notes2);

A Phrase that starts later

> Phrase phr2 = new Phrase("Phrase 2",4.0,JMC.FLUTE);
> phr2.addNoteList(notes2);
> View.notate(phr2)

Adding parts into phrases
(Wrong way first)

> Part part1 = new Part();
> part1.addPhrase(phr);
> part1.addPhrase(phr2);
> View.notate(part1);

Kinda lost the phrase distinctions.

Building Parts and Scores

> Part partA = new Part("Part A",JMC.PIANO,1)
> partA.addPhrase(phr);
> Part partB = new Part("Part B",JMC.SAX,2)
> partB.addPhrase(phr2);
> Score score1 = new Score("My Two Part Score");
> score1.addPart(partA);
> score1.addPart(partB);

Viewing the Score

> View.notate(score1);
Imports and some private data

```java
import jm.music.data.*;
import jm.JMC;
import jm.util.*;
import jm.music.tools.*;

public class AmazingGraceSong {
    private Score myScore = new Score("Amazing Grace");

    public void fillMeUp() {
        myScore.setTimeSignature(3, 4);
        double[] phrase1data = {JMC.G4, JMC.QN, JMC.C5, JMC.HN, JMC.E5, JMC.EN, JMC.C5, JMC.EN, JMC.E5, JMC.HN, JMC.D5, JMC.QN, JMC.C5, JMC.HN, JMC.A4, JMC.QN, JMC.G4, JMC.HN, JMC.G4, JMC.EN, JMC.A4, JMC.EN, JMC.C5, JMC.HN, JMC.E5, JMC.EN, JMC.C5, JMC.EN, JMC.E5, JMC.HN, JMC.D5, JMC.EN, JMC.E5, JMC.HN, JMC.D5, JMC.QN, JMC.C5, JMC.HN, JMC.A4, JMC.QN, JMC.G4, JMC.HN, JMC.G4, JMC.EN, JMC.A4, JMC.EN, JMC.C5, JMC.HN, JMC.E5, JMC.EN, JMC.C5, JMC.EN, JMC.E5, JMC.HN, JMC.D5, JMC.EN, JMC.E5, JMC.HN, JMC.D5, JMC.QN, JMC.C5, JMC.HN, JMC.A4, JMC.QN, JMC.G4, JMC.HN, JMC.G4, JMC.EN, JMC.A4, JMC.EN, JMC.C5, JMC.HN, JMC.E5, JMC.EN, JMC.C5, JMC.EN, JMC.E5, JMC.HN, JMC.D5, JMC.EN, JMC.E5, JMC.HN, JMC.D5, JMC.QN, JMC.C5, JMC.DHN};

        Phrase myPhrase = new Phrase();
        myPhrase.addNoteList(phrase1data);
        myPhrase.addNoteList(phrase2data);
        // create a new part and add the phrase to it
        Part aPart = new Part("Parts", JMC.FLUTE, 1);
        aPart.addPhrase(myPhrase);
        // add the part to the score
        myScore.addPart(aPart);
    }

    public void showMe() {
        View.notate(myScore);
    }
}
```

Amazing Grace

```java
AmazingGraceSong song1 = new AmazingGraceSong();
song1.fillMeUp();
song1.showMe();
```

Filling the Score

Each array is note, duration, note, duration, etc.

I broke it roughly into halves.

The Organization of JMusic Objects

- **Score**: timeSignature, tempo &
- **Part**: Instrument &
  - Phrase: startingTime &
    - Note: (pitch, duration)
  - Phrase: startingTime &
    - Note: (pitch, duration)

Thought Experiment

- How are they doing that?
- How can there be any number of Notes in a Phrase, Phrases in a Part, and Parts in a Score?
  - (Hint: They ain’t usin’ arrays!)
How do we explore composition here?

- We want to quickly and easily throw together notes in different groupings and see how they sound.
- The current JMusic structure models music.
  - Let’s try to create a structure that models thinking about music as bunches of riffs/SongElements that we want to combine in different ways.

Version 1: Notes in an array

- Let’s just put notes of interest (for now, just random) in an array.
- We’ll traverse the array to gather the notes up into a Phrase, then use View to notate the Phrase.

Using an array to structure Notes

```java
> Note[] someNotes = new Note[100];
> for (int i = 0; i < 100; i++)
>     {someNotes[i]= new Note((int)(128*Math.random()),0.25);}
> // Now, traverse the array and gather them up.
> Phrase myphrase = new Phrase()
> for (int i=0; i<100; i++)
>     {myphrase.addNote(someNotes[i]);}
> View.notate(myphrase);
```

Critique of Version 1

- So where’s the music?
  - 100 random notes isn’t the issue.
  - It’s that we don’t think about notes as just one long strand.
- Where are the phrases/riffs/elements?
  - We just have one long line of notes.
- How do we explore patterns like this?
  - insertAfter and delete are just as hard here as in sampled sounds!

Version 2: Using a linked list of song elements

- Let’s re-think Amazing Grace as a collection of elements that we can shuffle around as we’d like.
- We can make any element follow any other element.

What’s in each element?

AmazingGraceSongElement

- **KNOWS**: it’s Part and what comes next
- **CAN DO**: filling itself from the first or second phrase (with a given start time and instrument), setting the next one, getting the next one, and showing (notating) myself and all others.
What that would look like to use it

Welcome to DrJava.
> import jm.JMC;
> AmazingGraceSongElement2 part1 = new AmazingGraceSongElement2();
> part1.setPhrase(part1.phrase1(),0.0,JMC.FLUTE);
> AmazingGraceSongElement2 part2 = new AmazingGraceSongElement2();
> part1.getEndTime();
> part2.setPhrase(part2.phrase2(),22.0,JMC.PIANO);
> part1.setNext(part2);
> part1.showFromMeOn();

What’s going on here?

AmazingGraceSongElement part1

myPart: Filled with phrase1 (flute)
next: part2

AmazingGraceSongElement part2

myPart: Filled with phrase2 (piano)
next: null

Adding a third part

> AmazingGraceSongElement2 part3 = new AmazingGraceSongElement2();
> part3.setPhrase(part3.phrase1(),0.0, JMC.TRUMPET);
> part1.setNext(part3);
> part3.setNext(part2);
> part1.showFromMeOn();

What’s going on here?

AmazingGraceSongElement part1

myPart: Filled with phrase1 (flute)
next: part3

AmazingGraceSongElement part2

myPart: Filled with phrase2 (piano)
next: null

AmazingGraceSongElement part3

myPart: Filled with phrase3 (trumpet)
next: part2

part1.showFromMeOn();

Now has three parts
Introducing the **Linked List**

- A linked list is information broken into smaller pieces, where each piece knows the next piece, but none other.

Another example of a linked list

- Non-linear video editing (like in iMovie)
  - You have a collection of video clips (information)
  - You drag them into a timeline.
  - Each clip still doesn’t know all clips, but it knows the next one.

Why use linked lists versus arrays?

- Just two reasons now, more later:
  1. Can grow to *any* size (well, as long as memory permits)
     - Just create a new element and poke it into the list.
  2. MUCH easier to insert!
     - Look at how easily we put part3 between part1 and part2.

Implementing AmazingGraceSongElement2

```java
import jm.music.data.*;
import jm.JMC;
import jm.util.*;
import jm.music.tools.*;

class AmazingGraceSongElement2 {
    private AmazingGraceSongElement2 next;
    private Part myPart;

    // When we make a new element, the next part is empty, and ours is a blank new part
    public AmazingGraceSongElement2(){
        this.next = null;
        this.myPart = new Part();
    }

    // setPhrase takes a phrase and makes it the one for this element
    // at the desired start time with the given instrument
    public void setPhrase(Phrase myPhrase, double startTime, int instrument) {
        //Phrases get returned from phrase1() and phrase2() with default (0.0) startTime
        // We can set it here with whatever setPhrase gets as input
        myPhrase.setStartTime(startTime);
        this.myPart.addPhrase(myPhrase);
        this.myPart.setInstrument(instrument);
    }
}
```

Don't get hung up on these details—this is just manipulating the JMusic classes so that we can store the information we want.
The Phrases

```java
static public Phrase phrase1() {
    double[] phrase1data = {
        JMC.G4, JMC.QN, JMC.C5, JMC.HN, JMC.E5, JMC.EN,
        JMC.C5, JMC.EN, JMC.E5, JMC.HN, JMC.D5, JMC.QN,
        JMC.C5, JMC.HN, JMC.A4, JMC.QN, JMC.G4, JMC.HN,
        JMC.G4, JMC.EN, JMC.A4, JMC.EN, JMC.C5, JMC.EN,
        JMC.E5, JMC.HN, JMC.D5, JMC.EN, JMC.E5, JMC.EN,
        JMC.G5, JMC.DHN
    };
    Phrase myPhrase = new Phrase();
    myPhrase.addNoteList(phrase1data);
    return myPhrase;
}

static public Phrase phrase2() {
    double[] phrase2data = {
        JMC.G5, JMC.HN, JMC.E5, JMC.EN, JMC.G5, JMC.EN,
        JMC.G5, JMC.HN, JMC.E5, JMC.EN, JMC.C5, JMC.EN,
        JMC.E5, JMC.HN, JMC.D5, JMC.QN, JMC.C5, JMC.HN,
        JMC.A4, JMC.QN, JMC.G4, JMC.HN, JMC.G4, JMC.EN,
        JMC.A4, JMC.EN, JMC.C5, JMC.HN, JMC.E5, JMC.EN,
        JMC.C5, JMC.EN, JMC.E5, JMC.HN, JMC.D5, JMC.QN,
        JMC.C5, JMC.DHN
    };
    Phrase myPhrase = new Phrase();
    myPhrase.addNoteList(phrase2data);
    return myPhrase;
}
```

Static? This means that we can actually access them without any instances. Is that useful here?
Well, not yet...

Handling the linked list

```java
// Here are the two methods needed to make a linked list of elements
public void setNext(AmazingGraceSongElement2 nextOne) {
    this.next = nextOne;
}
public AmazingGraceSongElement2 next() {
    return this.next;
}
```

Controlling access: An accessor method

```java
// We could just access myPart directly
// but we can CONTROL access by using a method
// (called an accessor)
private Part part() {
    return this.myPart;
}
```

A little object manipulation

```java
// Why do we need this?
// If we want one piece to start after another, we need
// to know when the last one ends.
// Notice: It's the phrase that knows the end time.
// We have to ask the part for its phrase (assuming only
// one) to get the end time.
public double getEndTime() {
    return this.myPart.getPhrase(0).getEndTime();
}
```

The Key Part

```java
// Start from this element (this)
AmazingGraceSongElement2 current = this;
// While we're not through...
while (current != null) {
    // Set the channel, increment the channel, then add it in.
    current.setChannel(channelCount);
    channelCount += 1;
    myScore.addPart(current.part);
    // Now, move on to the next element
    current = current.next();
}
// At the end, let's see it!
View.notate(myScore);
```
**Step 1:**
// Start from this element (this)
AmazingGraceSongElement current = this;

**Step 2:**
// While we're not through...
while (current != null)
{  //BLAH BLAH BLAH
– PROCESS THIS PART

**Step 3:**
// Now, move on to the next element
current = current.next();

**Step 4:**
// While we're not through....
while (current != null)
{  //BLAH BLAH BLAH - PROCESS THIS PART

**Step 5:**
// Now, move on to the next element
current = current.next();

**Step 6:**
// While we're not through....
while (current != null)
{  //BLAH BLAH BLAH - PROCESS THIS PART
Step 7: // Now, move on to the next element
current = current.next();
};

T...
Critique of Version 2

- Lovely structuring of data, but just how much can one do with two parts of Amazing Grace?
  - We need the ability to have a library of phrases
- But what does the ordering mean? What if we had gone part1->part2->part3 instead?
  - What should the order encode?
  - Right now, it encodes nothing.
- When we’re exploring music, do we really want to worry about instruments and start times for every phrase?