INFORMATICA MUSICALE

UNIVERSITA' DEGLI STUDI DI CATANIA
DIPARTIMENTO DI MATEMATICA E INFORMATICA
LAUREA TRIENNALE IN INFORMATICA
A.A. 2020/21
Prof. Filippo L.M. Milotta

ID PROGETTO: 03

TITOLO PROGETTO: MIDI: programmare e comporre

AUTORE: Blanco Francesco Giulio

Indice

1. Obiettivi del progetto

0. Introduzione
    a. Elementi di Teoria Musicale
        i. Le note musicali
        ii. Il rigo musicale

1. Imparare a scrivere brani musicali in Java
    a. Il protocollo MIDI con approccio top-down
        i. Libreria offerta da Java: javax.sound.midi
        ii. Panoramica delle primitive della libreria
            - Giochiamo con i valori dei parametri per capirne il funzionamento
        iii. Problemi dell'uso di Java per la composizione musicale
            - Ricerca di un programma user-friendly per la composizione musicale

2. Un nuovo strumento compositivo: *Finale*
    a. Panoramica di *Finale*
        i. Come lavorare su Finale
        ii. Gli strumenti offerti dalla piattaforma

3. Parte finale**:** i risultati del progetto
    a. Programmare e comporre
    b. Due strumenti compositivi
        i. La scarsa scalabilità del codice Java
        ii. Il crollo della complessità della scrittura musicale su Finale
    c. Ultimo passo: usando i due strumenti posso ottenere un risultato equivalente, nonostante la diversa complessità?
        i. Posso aprire un brano MIDI scritto in Java con Finale?

2. Metodo Proposto e Riferimenti Bibliografici

0. Introduzione

La seguente introduzione ha lo scopo di presentare il pentagramma o rigo musicale e di introdurre alcuni concetti importanti (che torneranno utili nella programmazione) per la comprensione del progetto.

Le note musicali sono segni grafici mirati a rappresentare concettualmente e graficamente dei suoni (toni puri). Esse sono sette e hanno i nomi: do, re, mi, fa, sol, la, si. Le note differiscono fra loro per altezza (parametro legato alla frequenza del suono). La "distanza" fra una nota e l'altra (che è un delta in frequenza) è misurata, musicalmente, in "toni". Tra do e re, re e mi, fa e sol, la e si la distanza è di un tono. Tra mi e fa, si e do la distanza è di un semitono (1/2 tono). Possono essere definite delle note "intermedie" fra le note che hanno distanza 1 tono (rappresentate dai tasti neri del pianoforte). In totale, dunque, le note di una scala (da un do al do successivo) sono 12. Posso rappresentare le note in un "grafico" detto rigo musicale.

Sul rigo musicale, o pentagramma, posso collocare le note musicali secondo il seguente schema (figura 1):

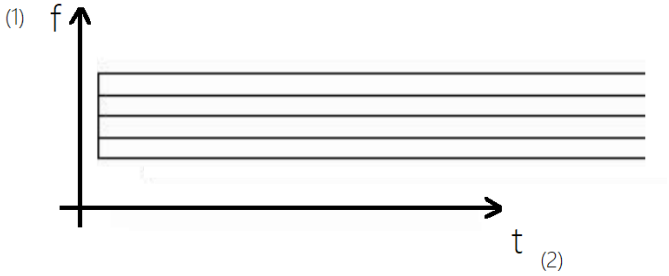

Figura 1

(1) Altezza. Più la nota musicale è alta (ovvero più la frequenza è elevata) più verrà collocata in alto sul pentagramma (una nota può essere collocata su una "linea" o su uno "spazio"). L'altezza va "misurata" a partire da un tono puro ben definito. Per farlo, si usano delle figure note come "chiavi musicali" (la figura 2 ne riporta due esempi). Scelta la chiave, essa va posizionata ad ogni inizio di rigo musicale.

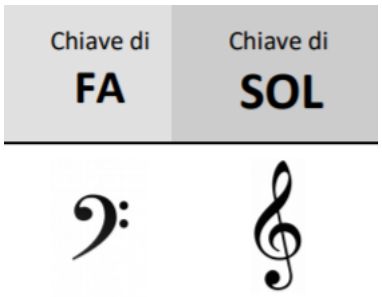

Figura 2: la chiave di fa e la chiave di sol

(2) Durata. L'asse delle ascisse segna lo scorrere del tempo musicale*. La durata di una nota si denota dal suo simbolo (figura 3), anche se esistono delle figure (es. punto di valore, che, posto di fianco alla nota, ne aumenta la durata di metà del suo valore) che ne possono alterare durata/modo di esecuzione. Esistono anche figure di silenzio (pause) di diversa durata.

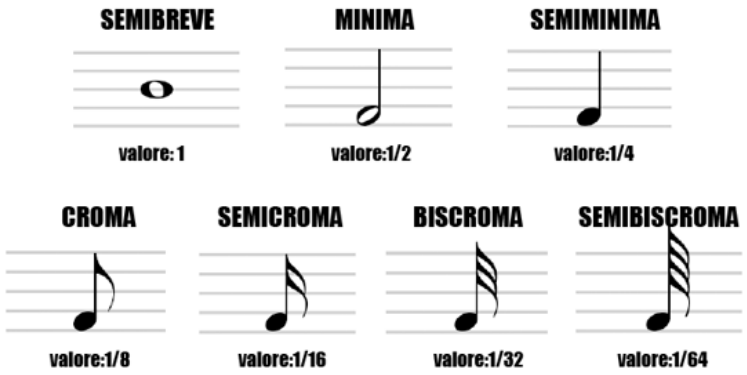

Figura 3

*il tempo musicale si misura in BPM (battiti per minuto). Non è un'unità di misura omogenea al tempo bensì alla frequenza.

Per maggiore chiarezza, dei testi di immediata consultazione di teoria musicale potrebbero essere questi: (non necessari per la comprensione del progetto)

1. http://gozzi-olivetti.org/attachments/article/270/TEORIA%20MUSICALE.pdf
   (livello di complessità basso, alta comprensibilità, pochi formalismi).
2. https://www.scuoladimusicacluster.it/images/Libro_Teoria_musicale_2018-19_(2%5Eed.).pdf
   (livello intermedio, maggiori formalismi, alta accuratezza).

1. Brani musicali in Java

Vogliamo scrivere un piccolo brano musicale utilizzando la libreria Java javax.sound.midi. Per farlo, è necessario utilizzare un approccio di tipo bottom-up, ovvero bisogna prima comprendere il protocollo MIDI e come si traduce l'uso del protocollo nelle primitive della libreria offerta da Java. Noi vogliamo però usare un approccio di tipo top-down, cioè comprendere il MIDI a partire dal codice Java che ho scritto.

Programmando, basta conoscere solo le note e non la loro rappresentazione sul rigo musicale. Tuttavia tali note (che noi scriveremo come degli "eventi" del protocollo) saranno poi riportate (tradotte) con un altro strumento (Finale, di cui verrà trattato in seguito) sul rigo musicale.

Il MIDI in Java

Java mette a disposizione una serie di Classi, metodi, Interfacce, Eccezioni per gestire, scrivere, modificare e suonare partiture in MIDI. Seguendo un approccio top-down, ricaviamo alcune caratteristiche del protocollo dalle primitive messe a disposizione dal linguaggio. L'ordine delle primitive è lo stesso ordine logico che andrebbe seguito per programmare in Java sfruttando il protocollo MIDI.

a. Ottenere un sequencer

```
// Get default sequencer, if it exists

        Sequencer sequencer = getSequencer();
```

//Un sequencer Midi è un dispositivo hardware, o più spesso oggi un software, che consente di registrare e modificare una performance musicale.

//Con registrare, non si intende la registrazione di suoni. Il sequencer, infatti, registra le note musicali e una serie di informazioni aggiuntive che sono i loro attributi.

//Esso traduce determinati stimoli (musica) in parametri.

b. *Modifica del tempo (espresso in BPM)*

//I parametri possono poi essere modificati

```
sequencer.setTempoInBPM(164.0f);
```

//Cambiamo il tempo

```
//sequencer.setTempoInBPM(144.0f);

//sequencer.setTempoInBPM(240.0f);
```

//Accetta anche valori molto elevati

```
//sequencer.setTempoInBPM(500.0f);

//sequencer.setTempoInBPM(2000.0f);
```

c. *Ottenere una sequenza*

```
seq = new Sequence(Sequence.PPQ, ticksPerQuarterNote);
```

//PPQ=0.0f -> "the resolution is expressed in pulses (ticks) per quarter note"

//ticksPerQuarterNote è settato a 4

// => Il tempo è 4/4. (ticks al numeratore, espresso in quarti)

d. *Setting generale*

```
private void setMidiEvents(Track track) {
```

//Questo metodo è diviso in 3 sezioni:

//Setting di:

//1. Strumenti musicali

//2. parametri (tempo, canali, nota di apertura..)

//3. note e partitura

e. *Setting degli strumenti musicali*

//1. Voglio creare un brano con tre strumenti.

```
//il primo sarà pianoforte, di default.
//il terzo saranno percussioni, che sono automaticamente settate al canale 10.
//il secondo è uno strumento che decido con i seguenti comandi:
        ShortMessage sm = new ShortMessage( );
//devo inviare uno shortmessage per cambiare il parametro strumento.
        int instrument = 26;
//facendo delle prove, associo un codice al suono di un possibile strumento…
//25 = chitarra acustica
//26 = chitarra classica
//16 = organo
//50 = violino
//57 = tromba
//…
//[…]
//il seguente metodo va collocato in un blocco try-catch
        sm.setMessage(ShortMessage.PROGRAM_CHANGE, 2, instrument, 0); //2 ==> is the channel 1.
//PROGRAM_CHANGE è una costante e vale 192 e indica il cambio dello strumento nello ShortMessage
//qui posso settare altri canali con altri strumenti se volessi aggiungere altri strumenti suonati al mio brano
        track.add(new MidiEvent(sm, 0));
```

f. Setting di tempo, canali, pitch di base

```
//2. Setting dei parametri
//Cambio i parametri per vedere cosa accade
        int channel = 1;
        //int channel = 3;
//nel metodo getMidiInputData() posso impostare lo strumento relativo al corretto canale
        int velocity = 64;
        //int velocity = 128; //out of range!
        int note = 60;     //note = pitch. Indica l'altezza di una nota. 128 valori possibili:
```

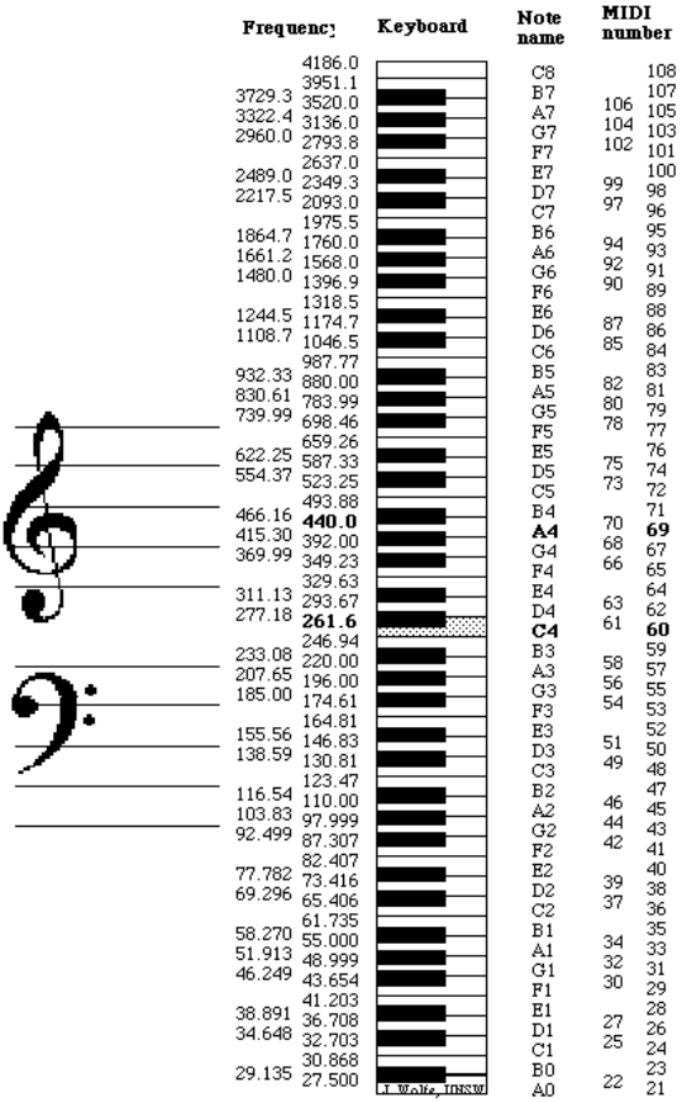

Figura 4: note in MIDI. Va considerato l'asse y come asse delle frequenze (le note variano per altezza => frequenza).

```
//int note = 60+12+12;
```

//Cosa accade? Aggiungendo 12 passo all'ottava successiva, essendo 12 i semitoni di una scala:

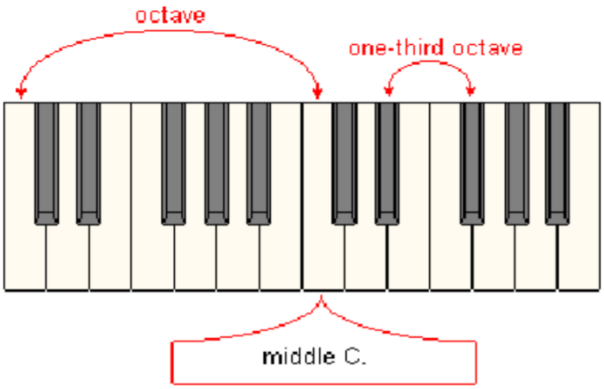

Figura 5

```
        int tick = 0;
        //int tick = 60;
//[…]
//primo strumento musicale
//per il pianoforte lascio i parametri di default (che ho settato prima)
//[…]
//secondo strumento musicale
//inserisco un secondo strumento musicale
//inizio accordi per l'intera durata del brano
//devo fare in modo che tale partitura venga suonata in contemporanea alla prima
        tick = 0;
        note = 60 +12; //do = 0; sol= +7; mi = +4; ottava superiore +12;
//in questo caso cambia solo il tipo di percussione utilizzata
        channel=2;  //il channel che ho riservato allo strumento
//[…]
//terzo strumento musicale
//inserisco le percussioni
         tick = 0;
        note = 60;
 //In questo caso cambiare “note” significa cambiare percussione e non cambiare nota
//(gli strumenti a percussione, in generale, non possono cambiare la nota suonata)
        channel=9;  //ho messo 9. Perché?
//Il channel 10 è automaticamente riservato alle percussioni, come si evince dalla seguente pagina:
//https://en.wikipedia.org/wiki/General_MIDI#Piano
//[…]
```

g. Scrittura della partitura

```
//3. partitura
//add midi Events
//inizio primo motivo
        addMidiEvent(track, ShortMessage.NOTE_ON, channel, note, velocity, tick);
         tick+=3;
```

```
            addMidiEvent(track, ShortMessage.NOTE_OFF, channel, note, 0, tick);
        tick+=1;
            addMidiEvent(track, ShortMessage.NOTE_ON, channel, note - 2, velocity, tick);
        tick+=3;
            addMidiEvent(track, ShortMessage.NOTE_OFF, channel, note - 2, 0, tick);
        tick+=1;
            addMidiEvent(track, ShortMessage.NOTE_ON, channel, note - 4, velocity, tick);
        tick+=3;
            addMidiEvent(track, ShortMessage.NOTE_OFF, channel, note - 4, 0, tick );
        tick+=1;
            addMidiEvent(track, ShortMessage.NOTE_ON, channel, note - 2, velocity, tick);
        tick+=3;
            addMidiEvent(track, ShortMessage.NOTE_OFF, channel, note - 2, 0, tick);
        tick+=1;
//many other events…
```

h. Scrittura su file MIDI

```
//Scrittura su file midi => Piena portabilità (apertura del file in output in altri programmi…)
        File f = new File("output.mid");
//[…]
//il seguente metodo va inserito in un blocco try-catch.
        MidiSystem.write(seq,1,f);
```

Scarsa scalabilità del codice Java

Il brano suonato generando manualmente una serie di eventi MIDI in Java ha richiesto la scrittura di circa 350 righe di codice, pur essendo il brano di breve durata.

Esempio:

Supponiamo adesso di voler riarrangiare per un duo di chitarre un brano più complesso e lungo come il “Tico tico” a partire dalla versione di Paco de Lucia del 1969. Il brano consta di circa 85 battute, ognuna, in media, di 9 note (indipendentemente dalla loro durata). Il numero complessivo delle note nel brano è circa 9*85 = 765. Supponiamo adesso di voler scrivere manualmente tale brano in Java, generando manualmente ogni singolo evento MIDI *(già all’apparenza impresa titanica)*. Assumendo come indice di complessità il numero di righe di codice, lo stesso **aumenta di** 4*765 = **3060 righe di**

codice, dato che un evento MIDI in Java richiede 4 righe di codice per essere avviato (oppure 2, ma con una complessità crescente):

```
addMidiEvent(track, ShortMessage.NOTE_ON, channel, note, velocity, tick);
tick+=3;
addMidiEvent(track, ShortMessage.NOTE_OFF, channel, note, 0, tick);
tick+=1;
```

A questa difficoltà si aggiunge quella della gestione di figure musicali particolari, che hanno durate diverse da quelle standard. Ad esempio, a causa di note come quelle in figura (crome [1/2], semicrome [1/4], semicrome con punto di valore [1/2+1/4=3/4]), bisognerà non più usare un conteggio "a 1 a 1", cioè contare il tempo a ritmo delle semiminime, ma siamo costretti a contare "a ¼ a ¼", complicando enormemente l'uso del *tick* (scarsa *scalabilità*).

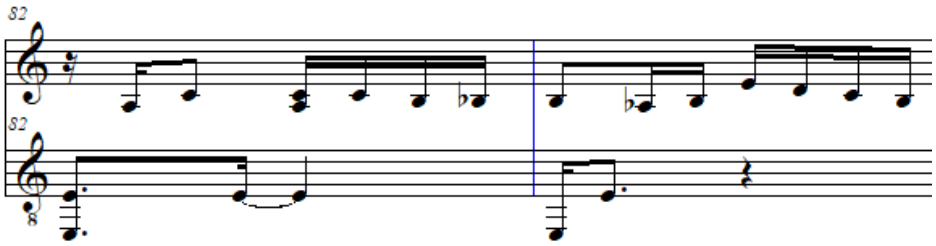

Figura 6: porzione della partitura "Tico Tico"

Risultato:

Il codice Java adoperato in questa sede non può essere assunto come strumento compositivo (in teoria potrebbe, ma risulterebbe estremamente sconveniente), ma principalmente didattico. Infatti l'uso delle librerie javax.sound.midi in Java risulta estremamente efficace per la comprensione del MIDI low-level.

Riferimenti (Oracle, Java documentation; the MIDI package)

❑ https://docs.oracle.com/javase/tutorial/sound/overview-MIDI.html

❑ https://docs.oracle.com/javase/tutorial/sound/MIDI-messages.html

❑ https://docs.oracle.com/javase/tutorial/sound/MIDI-seqmethods.html

❑https://stackoverflow.com/questions/35107057/java-midi-defaultsoundbank-playing-at-the-same-time-as-the-new-one

2. Un nuovo strumento compositivo: *Finale*

Finale è un programma per la scrittura di brani sul rigo musicale, dal pentagramma semplice al multiplo. Si tratta di uno strumento molto valido, usato anche dal noto compositore Justin Hurwitz (il compositore dietro il moderno musical *La La Land*).

Ne esistono varie versioni in commercio. Tuttavia, per la trattazione in questa sede, a meno di alcuni dettagli, possono essere adoperate tutte le versioni a partire dalla 2005 (potrebbe sembrare una versione datata, ma garantisce la piena portabilità: qualunque file generato da una versione *old* può essere aperto da una più recente mentre il contrario non può essere fatto). Se il lettore volesse dilettarsi con la realizzazione/trascrizione di alcuni brani, è disponibile una versione free di Finale chiamata Finale Notepad, ottima per la trattazione in questa sede.

Panoramica di *Finale*

Lo strumento si presenta estremamente intuitivo. All'avvio, scegliendo "impostazione guidata", possono essere selezionati tutti i parametri che la composizione dovrà avere: titolo (figura 7), tempo, scala, BPM, numero e tipo di strumenti musicali (figura 8) adoperati. I parametri possono comunque essere modificati in seguito.

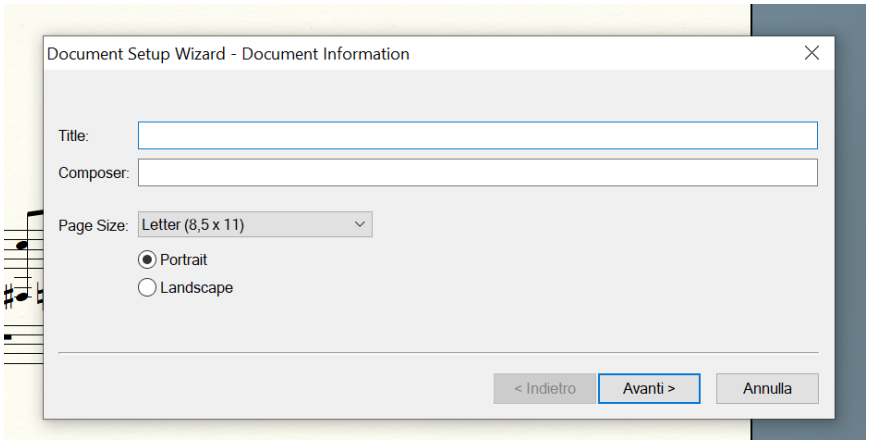

Figura 7: inserimento di titolo e compositore

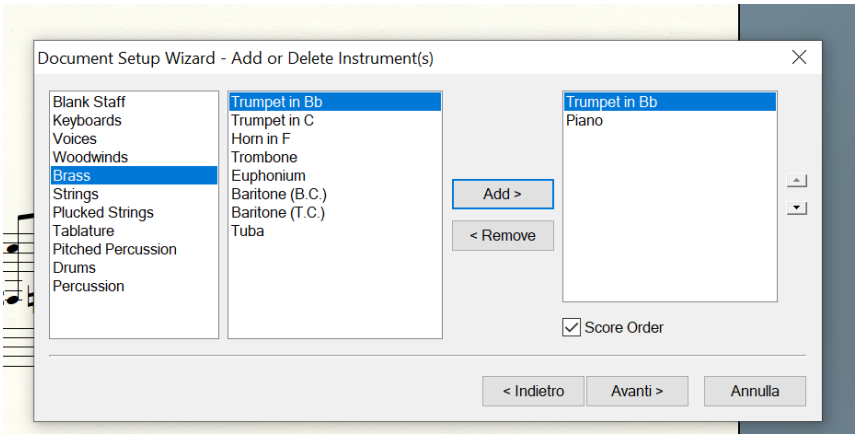

Figura 8: selezione degli strumenti musicali

Dopo che la prima fase è stata completata, la schermata (in visualizzazione pagina) è quella classica di un rigo musicale, avente battute vuote. Nella parte sinistra (o superiore) della schermata si possono selezionare le figure musicali (in questo caso riga 3 su 4 della figura 9), dopodiché con il mouse o con la tastiera del pc (frecce direzionali ed invio) possono essere posizionate sul pentagramma alla giusta altezza. Eventuali aggiustamenti (punto di valore, legatura di valore, punto coronato…) possono essere selezionati dalla stessa sezione o da menu appositi sulla parte superiore (in questo caso riga 3 su 4).

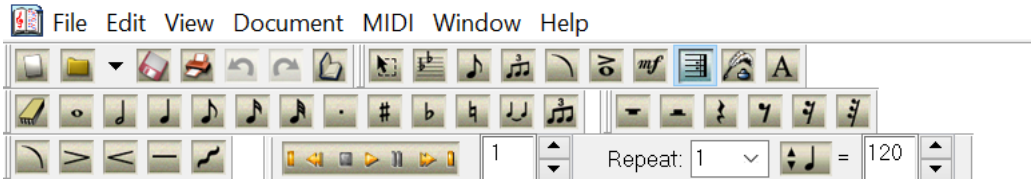

Figura 9: barre degli strumenti

Nei menu nella parte superiore dello schermo si possono selezionare vari “strumenti” (nel senso di “utensili”), che permettono di: aggiungere e rimuovere battute, cambiare la tonalità del brano, cambiare il tempo, aggiungere e rimuovere righi musicali, modificare le battute, aggiungere e rimuovere ripetizioni ed eventuali notazioni aggiuntive e molto altro (riga 2 su 4, figura 9).

Sempre nella parte superiore dello schermo è possibile cliccare sui tasti PLAY, STOP, PAUSE, NEXT, PREVIOUS (riga 4 su 4), che permettono l’ascolto della partitura. In genere, eventuali modifiche effettuate durante l’esecuzione o mentre essa è in PAUSE entrano in vigore dopo aver cliccato il pulsante STOP. Il sound ottenuto sarà quello degli strumenti musicali selezionati, dei quali è presente una ricca selezione. La pulsantiera di PLAY, STOP, PAUSE, NEXT, PREVIOUS è seguita dallo strumento di selezione della battuta. Infine, nella stessa riga, l’ultimo elemento serve a selezionare il tempo musicale (espresso in BPM).

Per trascrivere un brano di breve durata bastano tre minuti. Il rapporto fra le complessità C[] (C[Java]:C[Finale], in termini di ***throughput***) risulta molto minore di 1.

3. Risultati Ottenuti

Nelle precedenti pagine abbiamo:

1. generato **eventi MIDI in Java che abbiamo tradotto in una partitura** eseguita, testata e salvata su **un file MIDI.**
2. scritto uno spartito musicale su **Finale**, **abbassando notevolmente la complessità** rispetto alla generazione manuale degli eventi MIDI (codice Java).

Tuttavia, il nostro lavoro è stato gravoso poiché abbiamo lavorato usando un linguaggio di programmazione con una serie di primitive da questo offerte non solo per creare una partitura musicale, ma per emulare uno strumento compositivo su piattaforma. In altre parole, noi abbiamo non solo scritto la partitura, ma anche progettato una sorta di template modificabile che è uno strumento che emula Finale, sicuramente più ***interactive*** dal punto di vista del programmatore ma meno ***user-friendly***.

L'ultimo passo sarebbe:

3. dimostrare che **l'output dei due programmi sia identico** e che **l'output di uno possa essere usato come input dell'altro** (una sorta di "pipeline" concettuale).

In effetti è proprio così. Il file MIDI generato come output del programma Java può essere aperto e modificato su Finale (figura 10) e non genera problemi di portabilità.

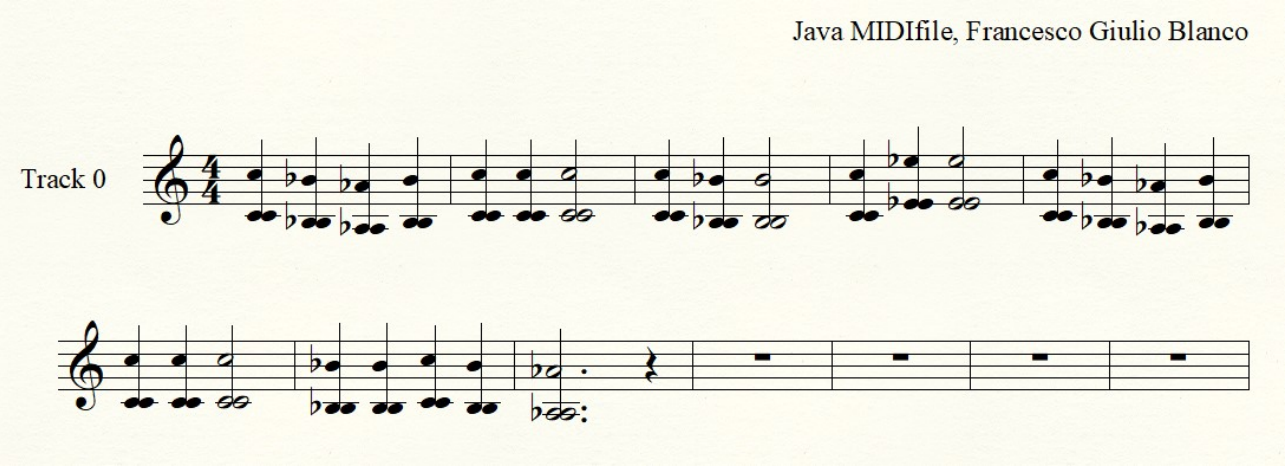

Figura 10: output del programma Java MidiPlayer.java aperto con Finale

Francesco Giulio Blanco