Lab 5: Interfaces and API: Yet another composer bot

Submit source code and UML diagram in PDF to Mooshak problem F at:

http://deei-mooshak.ualg.pt/~hdaniel

up to May 3, 2021

This lab assignment is composed by:

- the **source code**
- the **corresponding UML class diagram** in a *.pdf file.

When submitting to Mooshak include inside the folder with the source code a **uml.pdf** file with the UML class diagram.

Make sure that all the c**onstraints that specified below** and required for validation, are met before discussion.

How to check if the implementation is correct:

1)
Check that the output of the implementation is the expected according to the **Sample Cases**, presented in the specifications below.

2)
The output of the implementation can be passed as the input of the simple midi **Player** program distributed with these specifications (player.jar in player.zip), to play the riffs.

Extract the **player.zip**. Inside the extracted folder there is a jar file: **player.jar** and a folder **src** with the source code of the simple midi player.
Just run the jar from command line (or compile the source code in folder **src**) and pass in the first line the instrument to play (guitar or piano) and in the second line the output of the **composerBot** program:

Examples:

```
java -jar player.jar
piano
Gm4-1/8 Eo4-1/2 FM4-1/8 F4-1/8 D4-1/8 Gm4-1/2 C4-1/4 A4-1/8 Eo4-1/4 G4-1/8 A#4-1/2
```

```
java -jar player.jar
guitar
Bm4-1/4 B4-1/2 DM4-1/2 G4-1/2 GM4-1/8 C4-1/4 D4-1/8 D4-1/8 Bm4-1/2
```

The player can also be called form code, including the classes in the current project, and calling:

```
  play(new Sequence("C4-1/4 D4-1/4 E4-1/4"), 0, false, true);      //piano without chord arpeggio
  play(new Sequence("C4-1/4 D4-1/4 E4-1/4"), 1, true, true);       //guitar with chord arpeggio
```

Note: You can use whatever classes in Player, or modifications, to implement problem F.

Problem F: Yet another composer bot

1. Description

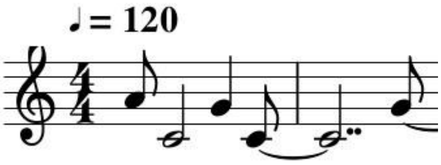
An uninspired musician, tired of spending hours squeezing his brain to figure out new licks and riffs, devised a(nother) way to automatically generates small sequences of notes: licks and small riffs.

Generated sequence of notes must be in tune and chords must sound good with solo notes, so the sequence cannot be just random.
He hopes that using a very over simplified version of basic western music theory, it is still possible to define simple strategies to get sequences of notes and chords that sound good when played together. This might be enough for a musician to be inspired to compose larger riffs an tunes … hopefully … anyway, below are his thoughts.

2. Musical notes

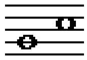
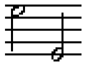
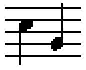
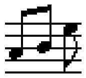
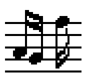
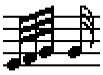
A musical note, or just note, sometimes also called **tone**, is composed by a **pitch** (a frequency in Hz) and a **duration**. This is the basic information needed to play a note. For a digital player it is needed also information on the timber of the instrument.

A note can also be represented by a **pitch class**, **octave** and **duration**. This is the information needed to write the note on a music staff, like the one on the right.
Pitch class is the name in the Chromatic scale: Do, re, mi, ... in Latin, or C, D E, … in English.
The **octave** gives a relative frequency to the same pitch class.
The **duration** is the length of the note since it is played until it is stopped playing.

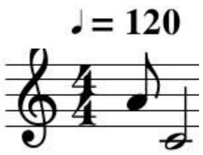


In a grand piano the middle octave is numbered 4. The A (or La) in the middle octave, or A4, have frequency 440Hz. One octave above, the frequency of the same pitch class, A5, is the double 880Hz. One octave below A3 pitch is 220 Hz.

Frequently use durations are the ones in the table below, expressed as fractions:

Note duration		Symbols	Duration
Semibreve	(whole note)		1
Minim	(half note)		1/2
Crotchet	(quarter note)		1/4
Quaver	(eighth note)		1/8
Semiquaver	(sixteenth note)		1/16
Demisemiquaver	(thirty-second note)		1/32

There must be a way to convert from relational note time to absolute time, in seconds. In a music staff there is an indication of the **Tempo**, a time value for a note duration in beats per minute (BPMs). It is presented at the beginning with the symbol of a note duration equal to an integer value. On the example staff on the right, we can see that the duration for a quarter note is 120 bpm. This means that in a minute, or 60 seconds, there will be 120 quarter notes. So, a quarter note duration is 0.5 seconds. Since all other durations are relative, they can be computed from this one. A half note duration is 1 second, an eighth note 0.25 seconds, and so on.



The composer bot does not have to be concerned with the frequency of the note or writing in a music staff. It will generate notes in the format:

<pitch class><octave>-<duration numerator>/<duration denominator>

Example:
A C# *(do sustenido)* with a duration of a quarter note, form octave 3, will be represented as the string:

C#3-1/4

Notes can have dynamic information too. For instance, the hardness of hitting a keyboard key, the hardness of plucking a string. This is addressed as the velocity of playing in Musical Instruments Midi Interface (midi), or in a music staff with the words *piano* (quiet) and *forte* (loud). This have impact in the loudness in which the note is heard.
This version of the composer bot will not implement this dynamic information.

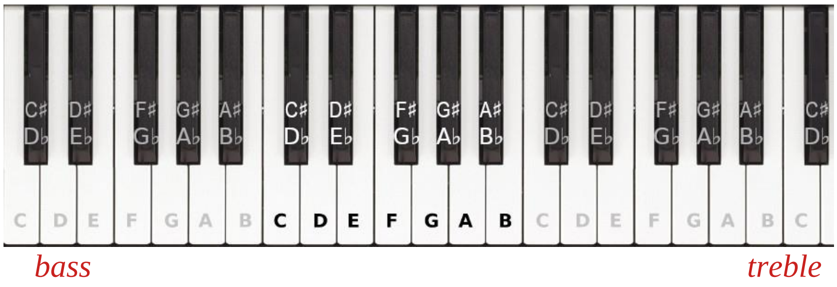
3. The chromatic scale

Western music is played with 12 **pitch classes**, sometimes called also **notes**, but without specifying octave and duration. This "notes" form the Chromatic scale: 7 natural notes and 5 sharps (or flats), each distancing one half step (in pitch) from the previous, and also next, pitch classes:

Table 1

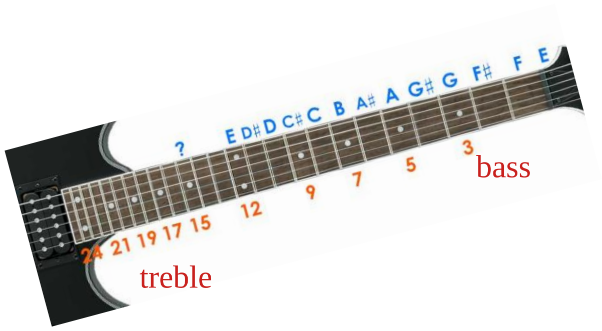
Latin	*Do*	*Do#*	*Re*	*Re#*	*Mi*	*Fa*	*Fa#*	*Sol*	*Sol#*	*La*	*La#*	*Si*
English	**C**	**C#**	**D**	**D#**	**E**	**F**	**F#**	**G**	**G#**	**A**	**A#**	**B**
		Db		***Eb***			***Gb***		***Ab***		***Bb***	

A sharp note (*sustenido*) is one half step up from the previous natural note, using equal temperament, where the 12 notes are at the same logarithmic distance.
A sharp can also be represented as a flat (*bemol*) note, one half step down from the next note. So, C# (C sharp) and Db (D *bemol* or D flat) are the same note between natural notes C and D. **For a question of simplification, we will define the next concepts only with flats.**



After 12 notes we have an **octave**. Octaves to the right of a keyboard repeat the same notes with higher frequency (treble), to the left with low frequency (bass). On the keyboard frequency (or pitch) grows to the right.

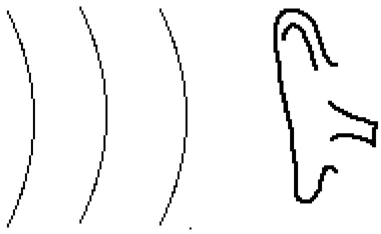
On the fretboard of a guitar, the notes have higher frequency (pitch) going down from the head to the body. This is due to the length of the string being played be smaller, when we fret near the body. being the vibrating string shorter, and shorter strings vibrate faster.



Also, the strings in a guitar have different thickness. Thicker strings vibrate slower than thin ones.

A piano has the same principle of short/long and thin/thick strings. When we press a key, a hammer hits a string inside the piano.

A faster vibration makes the air molecules vibrate faster. When they reach an eardrum, it also vibrates faster, giving a perception of a treble sound, higher in pitch. If the vibration is slower, it is perceived as a bass sound, lower in pitch.



4. Major and minor scales

Picking just random notes from the chromatic scale does not enforce harmony. If we need notes that sound well when played together, we need a specific set of notes from the Chromatic scale, but not all of them. We can use for instance pentatonic scales with 5 notes, heptatonic scales, with 7 notes, or other types of scales with more or less notes. Scales can also have a quality: Major, minor or others. Major scales sound happier and minor scales sound sombre or sadder. Actually, the scale used is what gives a flavour to the music being played.

The composer bot will use just **heptatonic** Major and minor scales. There are a Major and a minor scale for each of the pitch classes in the Chromatic scale, 24 scales in total. For instance, the C Major scale, the scale where the first pitch class, called **key** or **tonic**, is C, have the following order of notes (the white notes on a piano):

C D E F G A B

For each heptatonic scale there are also at least 7 chords that sound good when played together.

5. The circle of Fifths

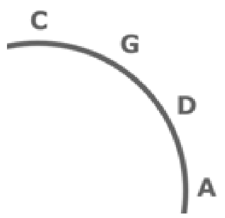
All the scales and corresponding chords can be computed from the “Circle of Fifths”. This circle orders notes in a circular fashion, distancing a Perfect Fifth from each order. A **Perfect Fifth** is an **interval** with 7 half steps apart. It is called perfect because it is considered perfectly consonant. The frequency ratio for the 2 notes in a perfect fifth is 3:2.

We can construct the circle of fifths from the chromatic scale, counting 7 half steps in a circular fashion, adding these pitch classes to the circle, until we reach all the 12. This way there will be no repetition of pitch classes in the circle.

Starting from C and counting 7 half steps we get G:

had

C# D# F# G# A#
C D E F G A B
0 7
14 21 ...



Now counting more 7 half steps, as if the scale is a circular array, we get D and after more 7 we get A, and so one. If we count 12 perfect Fifths from C and put then clockwise, equally distant, in a circle, starting with C at the top, we get the Circle of Fifths.

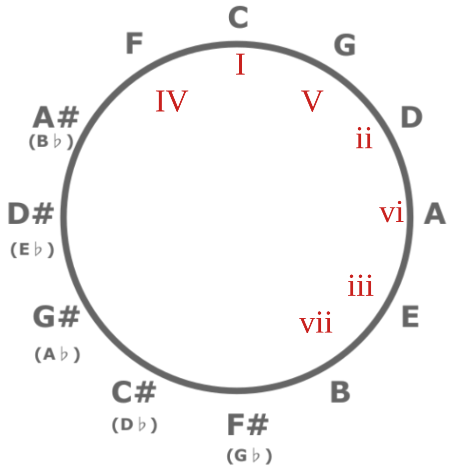
Having the circle of Fifths, the procedure to get **Major scales** already ordered form this circle is:

Starting at the key pitch class, get the pitch classes at the distance key + {0, 2, 4, 11, 1, 3, 5} positions clockwise, by this order.

If we start at C, we will get the C Major scale:

C D E F G A B

The relative order of pitch classes from the key usually are numbered with Roman numerals. For a Major scale they are the one in red on the circle.



To get E Major, start at E and get pitch classes at distances E key + { 0, 2, 4, 11, 1, 3, 5 }. We get the E Major scale already ordered:

E F# G# A B C# D#

To get **minor scales** the procedure is the same but the distance from the key pattern is different. For minor scales, the pattern is: key + {0, 2, 9, 11, 1, 8, 10}.
To get G minor scale, go to G and get the pitch classes at G key + {0, 2, 9, 11, 1, 8, 10} we get:

G A A# C D D# F

Using these two patterns we can get all the 12 Major and the 12 minor scales.

A Major scale have a relative minor scale that share the same pitch classes, but in a different ordering. C Major scale relative minor is A minor:

C Major:	C D E F G A B
A minor:	A B C D E F G

Scales that start with the same key, but have different notes are called parallel scales.

All these 24 scales are listed in the table below, defined using only naturals and sharps (#).

Table 2

The 12 Major scales	The 12 minor scales
C D E F G A B	C D D# F G G# A#
C# D# F F# G# A# C	C# D# E F# G# A B
D E F# G A B C#	D E F G A A# C
D# F G G# A# C D	D# F F# G# A# B C#
E F# G# A B C# D#	E F# G A B C D
F G A A# C D E	F G G# A# C C# D#
F# G# A# B C# D# F	F# G# A B C# D E
G A B C D E F#	G A A# C D D# F
G# A# C C# D# F G	G# A# B C# D# E F#
A B C# D E F# G#	A B C D E F G
A# C D D# F G A	A# C C# D# F F# G#
B C# D# E F# G# A#	B C# D E F# G A

6. Chords

The composer bot will use just Major, minor and diminished chords. These kind of chords are triads, each one is composed of 3 notes. For simplification we will consider that each note on the triad have the same duration.

Chords can be obtained directly from the chromatic scale, considering the root note and relative distance of the other notes to the root. The root note gives the name to the chord.
The relative distances in integer notation for a chord, depends on its quality, and are:

Major (M):	{ 0, 4, 7 }
minor (m):	{ 0, 3, 7 }
diminished (°) or (o):	{ 0, 3, 6 }

using two octaves of the Chromatic scale:

C C# D D# E F F# G G# A A# B C C# D D# E F F# G G# A A# B

To get A Major chord (symbol AM), the root node is A, the second note is 4 half tones to the right, which is C# and the third is E, more three half tones to the right., This way AM pitch classes are: A-C#-E.
For a minor and diminished chords the procedure is the same but it is used the patterns above.

The notation that the composer bot will use for a chord is like the notation for a note. There is no need to identify each of the notes, since all are relative to the root node, and we have already simplifyed that all have the same duration:

<chord><quality><octave of root>-<duration numerator>/<duration denominator>

F# Major chord on the third octave with duration a half note, is represented by:

F#M3-1/2

A minor chord on the second octave with duration a quarter note, is represented by:

Am2-1/4

B diminished chord on the fifth octave with duration a half note, is represented by:

B°5-1/2

7. Scale chords

To get the chords that sound good with a scale, go to the scale notes and get a chord for each note. As simple as that, the only caveat is the quality of the chord. It should be Major, minor or diminished? Well, there is a formula for that. Assuming, that each scale degree (the pitch classes) is identified by roman numerals, for instance:

C Major:	C	D	E	F	G	A	B
	I	ii	iii	IV	V	vi	vii°

The character case of the roman numeral indicates if it is a Major (uppercase) or minor (lowercase) chord. Diminished chord is vii°. The chords in the key of C Major are the ones below:

Major chords:	FM	CM	GM
minor chords:	Dm	Am	Em
diminished chords:	B°		

For a **minor scale**, the pattern is different, but the procedure is the same:

E minor:	E	F#	G	A	B	C	D
	i	ii°	III	iv	v	VI	VII

Now the diminished chord is ii°. The chords in the key of A minor are the ones below:

Major chords:	GM	CM	DM
minor chords:	Em	Am	Bm
diminished chords:	F#°		

8. Chord progressions

Chord progressions are sequences of chords to be played. They are associated to a scale and identified by putting its numbering in a sequence. Some popular chord progressions:

I–IV–V	rock, pop, country, blues, jazz, classical.
I–V–vi–IV	very popular among Billboard hot 100 hits
I–I–I–I	the 12 bar blues, and its variations. Not limited just to blues music.
IV-IV-I-I	
V-IV-I-V	
ii-V-I	jazz, rock, pop, R&B, country

For a Major scale and its relative minor scale, the chords are the same, with the same quality, however its roman numbering is different, making chords in the same progression different, and thus giving a different sound to the scale.
For a chord progression I-IV-V, the chords in the progression for C Major:

C Major:	C D E F G A B	are:	CM – FM – GM
	I ii iii IV V vi viiº		

The same chord progression, but now for a minor scale:

E minor:	E F# G A B C D	are:	Em – Am - Bm
	i iiº III iv v VI VII		

Note: To avoid character encoding incompatibilities with the diminished quality symbol (º), in the code should be used the lower case (o). All the examples and test samples below use (o) as the diminished quality symbol.

9. Composing strategies

For composing we will need to use **scales**, scale **notes**, **scale chords** and **chord progressions**.
We will use 2 different composing strategies: A and B.

Strategy A:
This is a chord only composing strategy. A scale is pseudo-randomly selected. From this scale a chord progression is also pseudo-randomly selected. Then also randomly a set of chords and durations are selected.
The simple pseudo-random generator will use a long integer, so define seed as: **long seed**. The following pseudo-code, very near Java code, represents the algorithm for **composing strategy A**:

**#1 Create a scale from the Circle of Fifths using patterns describe in: 5 The circle of Fifths
Scale quality: 0 is minor 1 is Major (even is minor, odd is Major)
Scale index is taken from Chromatic Scale (table 1), 0 is C, 1 is C#, … 11 is B**

```
long seed;                                              Example seed  = 1359233
int quality = (int) seed % 2                            quality       = 1359233 % 2 = 1
int scaleIdx = (int) seed % 12                          scaleIdx      = 1359233  % 12 = 5
seed =  seed / 3                                        seed          = 1359233 / 3 = 453077
Scale scale = CircleOfFifhts.getScale(scaleIdx, quality) Scale selected:   F Major
```

//#2 get all 7 notes and chords, for the scale, as described in: 6 Chords and 7 Scale chords

```
Notes[] notes      = scale.notes()              Scale notes:    F G A A# C D E
Chords[] chords = scale.chords()                Scale chords:   FM Gm Am A#M CM Dm Eo
```

//#3 Define chord progression

```
//how many chords in progression [3, 6]
noChords = (int) (seed % 3) + 3                 noChords      = (453077 % 3) + 3 = 5

//put in progression array the chords to use
for (int i=0; i<noChords; i++)
        int chordIdx = (int) seed % 7           chordIdx      = 453077 % 3 = 2
        seed =  seed / 3                        seed          = 453077 / 3 = 151025
        progression[i] = chords[chordIdx]       progression   = Am, …
                                                (…)
                                                progression   = Am FM CM Gm FM
                                                seed          = 1864
```

#4 generate lick/riff (strategy A)

```
reset seed to initial value                     seed          = 1359233
int progIdx = 0
int octave = 4
while (true)
        if(seed < 10) exit    //end generation when seed is small

        // Chord duration denominator {2, 4, 8}
        int duration = (int) Math.pow(2, (seed % 3)+1)     duration    = $2^{(1359233 \% 3) + 1}$ = 8
        seed =  seed / 3                                   seed        = 1359233 / 3 = 151025
        chord = progression[progIdx]                       chord       = progression[0] = Am
        if (++progIdx >= noChords) progIdx = 0             progIdx     = 1

        print or add chord to String with: <chord><octave><1/duration>        Am8-1/8
                                                                              (…)
```

Final output:
Am4-1/8 FM4-1/8 CM4-1/8 Gm4-1/4 FM4-1/4 Am4-1/4 FM4-1/4 CM4-1/2 Gm4-1/2 FM4-1/2 Am4-1/8

Strategy B:
This strategy's main difference from strategy A, is that between chords are also introduced solo notes. These notes are chosen also pseudo randomly. The difference is only in step **#4**:

Steps #1, #2 and #3 as in strategy A:

```
Example seed  = 45697852
quality       = 45697852 % 2 = 0
scaleIdx      = 45697852 % 12 = 4
seed          = 45697852 / 3 = 15232617
Scale selected:   E minor
scale notes:      E F# G A B C D
Scale chords:     Em F#o GM Am Bm CM DM

noChords      = (15232617 % 3) + 3 = 3
progression:      F#o CM Bm
```

#4 generate lick/riff (strategy B)

```
reset seed to initial value                     seed          = 45697852
```

```
int progIdx = 0
int octave = 4
while (true)
        // Chord duration denominator {2, 4, 8}
        int duration = (int) Math.pow(2, (seed % 3)+1)      duration = $2^{(45697852 \% 3) + 1}$ = 4
        seed =  seed / 3                                     seed     = 45697852 / 3 = 15232617
        chord = progression[progIdx]                         chord    = progression[0] = F#o
        if (++progIdx >= noChords) progIdx = 0               progIdx = 1

        print or add chord to String with: <chord><octave><1/duration>        F#o4-1/4

        //number of notes between chords: [0, 3]
        int noNotes = (int) seed % 4                         noNotes = 15232617 % 4 = 1
        for (int i = 0; i < noNotes; ++i)
                if(seed < 10) exit    //end generation when seed is small

                int noteIdx = (int) seed % 7                 noteIdx = 15232617 % 7 = 1
                // note duration denominator {2, 4, 8}
                dur = (int) Math.pow(2, (seed % 3)+1))       dur      = $2^{(15232617 \% 3) + 1}$ = 2
                seed =  seed / 3                             seed     = 15232617 / 3 = 5077539
                note = notes[noteIdx]                        note     = notes[1] = F#

                print or add note to String with: <note><octave><1/duration>   F#o4-1/4  F#4-1/2
                                                                               (…)
```

Final output:
F#o4-1/4 F#4-1/2 CM4-1/2 B4-1/2 Bm4-1/2 G4-1/8 F#o4-1/2 E4-1/2 E4-1/8 B4-1/8 CM4-1/8 C4-1/8 Bm4-1/4 F#o4-1/4

10. Task

Develop a composer bot that implements the composing strategies described above.

11. Constraints

The validation of an accepted submission depends on all the following requirements:

1) Client class must be called **ComposerBot**.

2) Use the design pattern “**Strategy**” to implement the 2 composing strategies.

3) Scales must be generated from the Circle of Fifths (see 5 The circle of Fifths). Define class **CircleOfFifths** and at least the method, used in both strategies above:

```
Scale getScale(int scaleIdx, int quality)
```

4) **Scale** class must have 2 methods to return all the notes and chords in the class, as was used in the composing strategies above:

```
Notes[] notes()
Chords[] chords()
```

5) Choose 1 class, except **ComposerBot**, and include in the comments, for all methods, pre and post conditions.

6) For the same class chosen on 5) define JUnit 5 tests for all methods.

7) The final UML class diagram, in *.pdf format, must be included in the source code folder

Suggestions for the implementation

Note that not all (the very oversimplified) part of the music theory described above is needed to implement the composer bot. It is just a very simple explanation on how things work together.

1) Define a class **Note** with instance variables: pitch class, octave and duration.
   Define a class Chord with instance variables: root, quality, octave and duration.

   You can take a look to the ones defined in the player.jar, as a starting point.
   Note that it is not needed to specify the 3 notes in each chord, just the name, quality, octave and duration, since the composer Bot will not play the chords.

   Any sequence of notes, a lick or a tune can be formed with objects of these 2 classes above.

2) Implement the class Circle of Fifths as a circular array of 12 notes. From there can be obtained any Major or Minor scale.

3) Scale chords with their quality (minor or Major) can be generated directly from the scale notes as shown in **7 Scale chords**.

4) Pay attention to the input and output. Create just the classes needed to do the task.

5) Check that Scale name and notes are correct for each sample, then check that the chord progression is also correct and finally the generated riff/lick.

12. Input

The input has 2 lines. The first line has only a character A or B, that specifies the composing strategy to use. The second line is a **long** integer used as seed for the riff/lick generator.

13. Output

The output has 4 lines. The first has the name of the scale, the second the notes on this scale, the third the chords in the progression and the fourth the generated sequence of notes and chords.

14. Operation samples

Below are some samples of expected operation:

Sample Input 0

```
B
12
```

Sample Output 0

```
C minor
C D D# F G G# A#
Gm Do Cm Cm
Gm4-1/2 Do4-1/4
```

Sample Input 1

```
B
759854
```

Sample Output 1

```
D minor
D E F G A A# C
Gm Eo FM
Gm4-1/8 Eo4-1/2 FM4-1/8 F4-1/8 D4-1/8 Gm4-1/2 C4-1/4 A4-1/8 Eo4-1/4 G4-1/8 A#4-1/2 FM4-
1/4
```

Sample Input 2

```
A
2342534
```

Sample Output 2

```
D minor
D E F G A A# C
Eo Dm FM Gm
Eo4-1/8 Dm4-1/4 FM4-1/4 Gm4-1/2 Eo4-1/2 Dm4-1/4 FM4-1/2 Gm4-1/2 Eo4-1/2 Dm4-1/8 FM4-
1/2 Gm4-1/4
```

Sample Input 3

```
A
1636322
```

Sample Output 3

```
D minor
D E F G A A# C
Dm FM A#M CM
Dm4-1/8 FM4-1/4 A#M4-1/4 CM4-1/4 Dm4-1/8 FM4-1/4 A#M4-1/2 CM4-1/4 Dm4-1/2 FM4-1/8
A#M4-1/2
```

Sample Input 4

```
B
672526734
```

Sample Output 4

```
F# minor
F# G# A B C# D E
C#m Bm Bm G#o AM
C#m4-1/2 C#4-1/8 B4-1/4 Bm4-1/2 G#4-1/8 A4-1/2 B4-1/2 Bm4-1/8 A4-1/8 F#4-1/2 F#4-1/4 -
G#o4-1/4 D4-1/8 AM4-1/4 C#m4-1/8 C#4-1/4 G#4-1/2 Bm4-1/8
```