INF-111 Travail pratique #2

Auteurs : Pierre Bélisle
Adaptation pour le cours INF111-04 par Mélanie Lord

Travail en équipe de 2

En avant la musique

Remerciement

Je tiens à remercier monsieur Frédérick Boulanger, du département d'informatique de Supélec en France, qui a mis en place tout le code pour produire des notes et qui nous a généreusement permis de l'utiliser.

1 Objectifs

- Familiarisation avec le langage Java et ses collections (ArrayList, Vector,…).
- Utilisation de classes et d'héritage.
- Pratique des commentaires Javadoc.

2 Description du problème : Faire jouer des sons

Tout le monde sait qu'un ordinateur peut produire des sons. Cependant, ce n'est pas si simple de reproduire exactement (ou même proche) celui qu'on veut entendre ou d'en entendre plusieurs à la fois.

Ce travail se veut un simple prétexte à l'utilisation de la programmation orientée-objet et des collections de Java. Il n'est aucunement notre intention de prétendre que les logiciels de musique fonctionnent ainsi, mais nous allons tout mettre en place pour reproduire des notes de musique et même des accords.

Il y aura peut-être parfois des petits écarts de son dû au parallélisme géré très sommairement dans ce projet.

3 Mandat

Il s'agit d'écrire une application Java avec une conception orientée-objet imposée qui permet de lire des accords d'une chanson dans un fichier texte et de les jouer.

Le code doit respecter les bonnes pratiques de programmation enseignées et obligatoirement respecter les consignes qui sont décrites tout au long de cet énoncé.

4 Un peu de théorie des ondes

Un son est une onde analogique que l'on peut transformer en signal numérique en une séquence de nombres binaires. Nous vous faisons grâce de toute la théorie physique associée à cela, et nous ne vous demanderons de compléter qu'une partie du code nécessaire qui provient de monsieur Boulanger. Les détails vous sont fournis plus bas.

5 Un petit retour sur la théorie musicale

Une gamme est constituée de notes qui sont situées à des intervalles comptés en ton. Dès qu'on a une gamme, nous pouvons constituer des accords dans cette gamme.

Un accord est constitué d'au moins une triade de notes jouées simultanément (on parle d'intervalle pour 2 notes jouées ensemble et ce n'est pas un accord). On peut ajouter des notes et en avoir plus que trois. Dans notre projet, nous utiliserons les accords à 3 et à 4 notes. En considérant que les notes sont numérotées, un accord est minimalement constitué des notes 1,3 et 5 de la gamme et nous ajouterons la 7e pour notre projet.

Par exemple pour la gamme de do majeur que vous connaissez :

C D E F G A B
1 2 3 4 5 6 7

L'accord de C est constitué des notes C E G jouées ensemble (1, 3, 5)
L'accord de C7 : C E G Bb (la 7e est diminuée d'un demi-ton, c'est la loi).
L'accord de Cm : C Eb G Bb
L'accord de Cm7 : C Eb G Bb

Tout cela pour votre culture personnelle, **le plus important à retenir c'est qu'il y aura 3 ou 4 notes dans les accords de notre projet**.

6 Conception

Voici les détails de la conception qui vous est imposée. Votre projet doit absolument suivre ces consignes. Tout le reste est à votre discrétion.

6.1 Les formes d'ondes

Un convertisseur analogue à digital transforme le signal électrique qui est un signal continu, en petites successions d'événements qui s'enchaînent à intervalles de temps réguliers. La fréquence d'échantillonnage est l'intervalle de temps entre deux échantillons[1]. Elle est exprimée en Herz [Hz]. Plus le son est aigu et plus la

[1] https://owl-ge.ch/travaux-d-eleves/2007-2008/article/qu-est-ce-qu-une-frequence-d-echantillonnage

fréquence d'échantillonnage doit être élevée. Au contraire, plus le son est grave et plus elle doit être faible.

Pour reproduire une note, il faudra simuler des formes d'ondes. Cela nécessite une fréquence et une fréquence d'échantillonnage. Il s'agira d'obtenir des échantillons particuliers de ces différentes formes d'ondes. Par exemple, la note A peut être reproduite avec une fréquence 440Hz et une fréquence échantillonnage de 20500. Tout cela, monsieur Boulanger s'en est occupé.

Quatre formes d'ondes seront suffisantes. C'est la forme de l'onde sur un oscilloscope qui lui a donné son nom : aléatoire, triangulaire, carrée et sinusoïdale. Nous vous demandons d'écrire la hiérarchie de classes, et le code nécessaire à la classe **Note** qui vous est fournie. C'est de votre responsabilité de minimiser la répétition de code.

6.1.1 Classe FormeOnde

Cette classe est la classe "parent" la plus haute de la hiérarchie. Elle a 2 attributs qui sont la **fréquence** et la **fréquence d'échantillonnage**.

Cette classe est immuable et n'a qu'un constructeur d'initialisation qui prend en paramètres deux valeurs (type double) pour initialiser ses deux attributs. Écrivez aussi les 2 accesseurs (*getters*).

Les classes FormeAleatoire, FormeCarree, FormeTriangulaire, et FormeSinusoidale), expliquées dans les sections suivantes, héritent (sont des enfants) de la classe FormeOnde. Pour ce faire, vous devez le spécifier dans l'entête de ces 4 classes avec le mot réservé "extends". Par exemple :

public class FormeCarree **extends** FormeOnde {...}

De plus, il vous faut savoir qu'une classe "enfant" est responsable de la construction de son parent. Pour ce faire, vous devrez appeler le constructeur de la classe parente dans le constructeur de chaque classe "enfant", à l'aide du mot réservé super. Par exemple, le constructeur de la classe FormeCarree ressemblerait à ceci (il n'y a rien d'autre à faire) :

```
public FormeCarre(double frequence, double freqEchantillon) {
        super (frequence, freqEchantillon);  / /appel du constructeur de la classe
                                              // parente
}
```

En ce qui concerne la classe FormeAleatoire, celle-ci n'a pas besoin d'une fréquence et d'une fréquence d'échantillonnage, et donc son constructeur ne prend aucun argument, et appelle le constructeur de sa superclasse avec 2 valeurs de 0 :

```
        super(0, 0);
```

En plus de leur constructeur, les classes "enfants" implémentent toutes la même méthode :

```
// Retourne le ième échantillon de la forme d'ondes.
public double echantillon (int i)
```

6.1.2 Classe FormeAleatoire

La méthode echantillon de cette classe doit retourner un nombre entre -1 et 1 choisi au hasard. L'échantillon est un nombre réel choisi au hasard dans [0.0, 1.0[puis étendu à l'intervalle [-1, 1[. Il suffit d'écrire 2 * Math.random() et de lui soustraire 1.

6.1.3 Classe FormeCarree

La méthode echantillon de cette classe doit retourner -1.0 ou 1.0. Utilisez le calcul suivant :

La **période** du signal est 1 / fréquence de l'onde

r est le reste de la division entre (a = i/fréquence d'échantillonnage), et la période. (Utilisez Math.IEEEremainder (a, période)).

Si **r** est plus petit que 0, on lui ajoute la période.

Retourner 1.0 si **r** est plus petit que la moitié de la période, sinon retourner -1.0

6.1.4 Classe FormeTriangulaire

La méthode echantillon de cette classe doit retourner dans son premier quart une valeur vers 1. Dans ses 2e et 3e quarts, cela descend de 1 vers -1 et dans son dernier quart, cela remonte vers 0. Utilisez le calcul suivant :

La **période** du signal est 1 / fréquence de l'onde

r est le reste de la division entre (a = i/fréquence d'échantillonnage), et la période. (Utilisez Math.IEEEremainder (a, période)).

Si **r** est plus petit que 0, on lui ajoute la période.

Si **r** est plus petit ou égal à la période sur 4, la valeur à retourner est égale à r * 4 / période.

Sinon si **r** est plus petit ou égal à 3 fois la période sur 4, **r** est diminué du quart de la période, et la valeur à retourner est égale à 1 - r * 4 / période.

Sinon, **r** est diminué de 3 fois la période / 4, et la valeur à retourner est égale à -1 + r * 4 / période.

6.1.5 Classe FormeSinusoidale

La méthode échantillon de cette classe doit retourner un échantillon d'une forme sinusoïdale. Utilisez le calcul suivant :

La **période** du signal est 1 / fréquence de l'onde

r est le reste de la division entre (a = i/fréquence d'échantillonnage), et la période. (Utilisez Math.IEEEremainder (a, période)).

Si **r** est plus petit que 0, on lui ajoute la période.

Retourner le sinus de 2 * ∏ * la fréquence de l'onde * **r**.

6.2 Test de son

Pour tester votre code, vous devez utiliser un canal qui permet de jouer une note. La classe AudioFormat de Java permet cela. Voici ce que vous devez importer :

```
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.AudioFormat;
import javax.sound.sampled.LineUnavailableException;
import javax.sound.sampled.SourceDataLine
```

Il s'agit d'obtenir une instance de la classe AudioFormat. Nous vous faisons grâce des détails, mais voici les valeurs à utiliser.

```
AudioFormat audioFmt =
        new AudioFormat(20500, 16, 1, true, true);
```

On a besoin d'une ligne audio. Une SourceDataLine est une sorte de canal appelée une ligne, qui permet de jouer un son. On envoie des notes par cette ligne et elles seront jouées. Voici le code pour ouvrir la ligne.

```
try {
        ligne = AudioSystem.getSourceDataLine(audioFmt);
        ligne.open(audioFmt);

} catch (LineUnavailableException e) {
        System.out.println("# Erreur : impossible de trouver une
                                ligne de sortie audio au format :");
        System.out.println("#         " + audioFmt);
}
```

Ensuite, il faut démarrer l'envoi des données. Pour cela nous allons l'exécuter dans un processus de la classe Thread. Cela démarrera la production du son dans un processus séparé de votre main(). Voici le code nécessaire :

```
Thread t = new Thread (new Runnable() {

        @Override
        public void run() {

                // Démarre la production de son
                ligne.start();

                // Joue la gamme de do majeure à une intensité de
                // 30% pendant une demie seconde.  Le 4 signifie
                // la gamme au centre du piano.  C3 est plus grave et
                // C5 est plus aigu.

                jouer (ligne, "C4",.3, 500);
                jouer (ligne, "D4",.3, 500);
                jouer (ligne, "E4",.3, 500);
                jouer (ligne, "F4",.3, 500);
                jouer (ligne, "G4",.3, 500);
                jouer (ligne, "A4",.3, 500);
                jouer (ligne, "B4",.3, 500);
                jouer (ligne, "C5",.3, 1000); // do plus aigu, 1 seconde
        }
});

t.start();
```

Il reste à écrire la méthode jouer utilisée dans le code précédent. Celle-ci reçoit une ligne type (SourceDataLine), une note (de type Sring, ex. : "C4"), une intensité (type double), et une durée (type double) en millisecondes. Elle joue la note sur la ligne avec ces 2 lignes de code.

```
Note n = new Note (note, duree);  //création d'une Note
n.jouer (ligne, intensite); //appel de la méthode jouer de la classe
                            //Note
```

Partie 1 : 30%
Écrivez un programme principal dans une classe nommée **TestJouerNotes** qui joue les notes de la gamme de do majeur (do, ré, mi, fa,.... si) en utilisant le code décrit précédemment et les classes Note, Notation_US (utilisée par la classe Note seulement), et vos classes de formes d'ondes écrites précédemment (utilisées par la classe Note seulement). Faites jouer chaque note 500 millisecondes, à une intensité de 0.3.

7 La classe Accord

Il s'agit ici d'écrire la classe **Accord**. Elle a 2 attributs qui sont le nom de l'accord (de type String), et une collection de Notes. Utilisez la classe ArrayList pour conserver les notes (de type Note) qui constituent l'accord. Cette collection n'aura que 3 ou 4 notes, mais pourrait éventuellement en avoir plus.

7.1 Le constructeur et un accesseur

Le constructeur reçoit en paramètres le nom de l'accord (pour initialiser l'attribut nom) ainsi qu'un tableau contenant des notes (de type Note[]). Il s'agit de prendre chaque note du tableau et de les ajouter à la collection (ArrayList).

Écrivez aussi l'accesseur (*getter*) pour l'attribut nom.

7.2 La méthode equals

La méthode equals doit redéfinir celle de la classe Object. Voici son entête, écrivez le reste qui dit que deux accords sont identiques, s'ils ont le même nom.

public boolean equals (Object accord)

7.3 La méthode Jouer

Vous devez maintenant permettre de jouer l'accord. Pour ce faire, utilisez le code de la partie un pour jouer des notes, à l'exception que vous jouez les notes qui sont dans la collection. Comme il faut jouer les notes en même temps (accord), il faut plusieurs lignes audios, chaque note devant être jouée sur sa propre ligne, envoyée dans son propre Thread.

Voici en pseudocode ce que la méthode doit réaliser :

Début

Déclarer et instancier un tableau avec autant de SourceDataLine qu'il y a de notes (il doit y avoir une ligne par Note).

Ouvrir les lignes

Pour chacune des notes i de la collection de notes

Retenir i dans une variable j (utile au Thread)[2]

```
Thread t = new Thread(new Runnable() {

    @Override
    public void run() {

        line[j].start();
        /*
          Il s'agit pour vous de jouer la note de l'accord
          qui se trouve à la position j dans votre
          collection.
        */
    }
});

// Démarrage du Thread.
t.start();
```

}

Fin

7.4 La méthode toString()

Redéfinissez la méthode toString pour qu'elle retourne une représentation de l'accord sous forme d'une chaine de caractères. Voici son entête (à respecter) :

```
public String toString()
```

La chaine retournée doit contenir le nom de l'accord, ainsi que chaque note (et sa durée) faisant partie de cet accord. Par exemple pour l'accord de C, on devrait obtenir une chaine qui, si elle est affiché, afficherait quelque chose du genre :

```
Nom de l'accord : C
Notes : C4 272.0 E4 272.0 G4 272.0
```

N'oubliez pas d'utiliser la méthode toString() de la classe Note, qui fait déjà une partie du travail.

---

[2] Comme vous le verrez si vous l'essayez, l'itérateur i est inutilisable dans le Thread (Local variable defined in an enclosing scope must be final or effectively final). D'où la nécessité de j.

8 Générateur d'accords

Pour tester votre classe Accord, nous avons créé un générateur d'accords. Cette classe permet d'obtenir des accords d'une durée fixée à l'instanciation. C'est important, car toutes les notes doivent être jouées exactement le même nombre de battements. Nous y reviendrons lorsque nous jouerons une pièce musicale.

Les accords qui sont générés sont les accords majeur, dominant 7, mineur et mineur 7, pour toutes les notes de la gamme non accentuées (sans bémol ni dièse). Leur nom est C, C7, Cm et Cm7 pour le do, D, D7, Dm et Dm7 pour le ré, etc.

Partie 2 : 30%

Il s'agit pour vous de générer des accords, et de les faire jouer. Écrivez un programme principal dans une classe nommé **TestJouerAccords** qui fait jouer les accords **C Dm Em F G Am B7** (1000 millisecondes pour chaque accord). Aucune optimisation n'est nécessaire ici, il s'agit de tester votre classe Accord et votre compréhension du générateur.

NOTE : Utilisez la méthode pause expliquée à la section 9.3 pour faire une pause (de 1000 millisecondes aussi) entre chaque accord joué

9 La classe PieceMusicale

Une pièce musicale est une suite d'accords joués à un certain rythme compté en battement par minute (bpm). Plus le nombre est gros, plus la pièce est rapide, et inversement. Chaque accord est joué pendant un certain nombre de battements qui peut être divisé en demi(.5), quart(.25), huitième(.125), etc.

Dans cette classe, vous avez 3 attributs : le nom de la pièce musicale, le rythme en bpm, et une collection d'accords. Vous utiliserez la classe **Vector** pour conserver les accords.

9.1 Le constructeur

Le constructeur reçoit le titre et le nombre de bpm (int). Il instancie la collection d'accords qui est vide pour l'instant.

9.2 L'ajout d'un accord

Écrivez la méthode **ajouterAccord** qui prend en paramètre un accord (type Accord) et l'ajoute à la collection privée (en fin de liste).

9.3 La méthode pause.

Nous vous offrons la méthode suivante qui permettra de jouer l'accord de la bonne durée. Appeler cette méthode entre chaque accord joué, avec la même durée que l'accord qui précède la pause.

```
/**
 * Permet de pauser l'application pour donner du temps à
 * un autre processus dans un univers multitâches.
 *
 * @param duree Le temps de la pause.
 */
private void pause (int duree) {

        try {
                Thread.sleep(duree);

        } catch (InterruptedException e) {

                e.printStackTrace();
        }

}
```

9.4 La méthode jouer

Cette méthode permet de jouer un accord à la fois, de la bonne durée. Il s'agit d'appeler la méthode jouer pour chaque accord de la collection et d'effectuer une pause de la durée de l'accord entre chaque.

9.5 La méthode toString()

Écrivez cette méthode, elle vous aidera au débogage. Il s'agit d'appeler en boucle la méthode toString de la classe Accord, et de les concaténer en sautant une ligne à tous les 4 accords. Par exemple pour l'accord de C d'un battement sur 220 bpm on devrait voir quelque chose du genre :

10 Fichier texte d'accords

Dans un fichier texte, vous pouvez écrire le titre d'une pièce musicale sur une ligne, le nombre de battements par minute sur la suivante, et ensuite 4 accords par ligne. Un accord étant suivi de sa durée (un nombre réel). Nous vous avons fourni au clair de la lune dans le fichier nommé « auClairDeLaLune.txt ».

Vous devez écrire la classe Fichier qui ne contient qu'une méthode publique permettant d'obtenir une pièce musicale dans un fichier sélectionné par l'utilisateur.

public static PieceMusicale obtenirChanson (String cheminFichier)

Veuillez utiliser la classe FileChooser de Java pour obtenir le chemin du fichier à lire, pour le passer ensuite en paramètre à la méthode obtenirChanson. Vous trouverez comment l'utiliser pour obtenir un nom de fichier sélectionné par l'utilisateur ici : :
https://docs.oracle.com/javase/tutorial/uiswing/components/filechooser.html

Notez que si le fichier contient des accords non jouables par notre application, elle ne fonctionnera pas, mais nous ne ferons rien pour empêcher cela. Seuls des accords majeurs, mineurs, dominants 7^e et mineurs 7^e sont acceptés.

Voici les grandes étapes en pseudoCode de la méthode obtenirChanson :

Début

Déclarer un objet de la classe PieceMusicale (à retourner)

Créer un objet de type File avec le fichier reçu en paramètre (voir java.io.File; encore un petit peu de recherche)

Essayer (try)

Déclarer et instancier un objet de la classe Scanner avec le fichier de type File (au lieu de System.in).

Déclarer et instancier un objet de la classe GenerateurAccord.

Instancier l'objet de la classe PieceMusicale.

Lire le titre de la pièce dans le fichier (nextLine)

Lire le nombre de bpm (nextInt)

Tant qu'il reste des accords à lire (hasNext)

Lire un accord (next) et sa durée (nextDouble)

Il faut convertir la durée par rapport au bpm. La durée réelle est : durée * (60000/bpm). Le 60000 vaut une minute en millisecondes.

Obtenir l'accord du générateur d'accords avec la bonne durée et l'ajouter à la pièce musicale.

Fin de la boucle

S'il y a une exception FileNotFoundException, affichez la trace de la pile système (printStackTrace).

Retourner la pièce.

Fin

Partie 3 : 40%

Il s'agit pour vous d'écrire un programme dans une classe nommée **JouerPieceMusicale** qui offre un menu de 2 options à l'utilisateur, à l'aide de JOptionPane.showInputDialog :

1) Sélectionner une pièce à écouter
2) Rejouer la dernière pièce écoutée

Lorsque l'utilisateur choisit l'option de sélection d'une pièce, vous devez obtenir la pièce à l'aide d'un objet de la classe Fichier.

Pour l'autre option, vous jouez simplement la pièce à l'aide de sa méthode jouer(). S'il n'y a aucune sélection préalable, le programme avise l'utilisateur qu'il doit sélectionner la pièce d'abord à l'aide de JOptionPane.showMessageDialog.

IMPORTANT :

Pour assurer la compatibilité de l'application avec Mac, ajoutez le code suivant dans votre classe JouerPieceMusicale :

```
import javax.swing.UIManager; //Dans le haut de votre classe
```

Méthode à ajouter :

```
public static void preparerPourMac() {

      try {
            UIManager.setLookAndFeel(
            UIManager.getCrossPlatformLookAndFeelClassName());

      } catch (Exception e) {
            e.printStackTrace();
      }
}
```

Au début de la méthode `main`, appelez la méthode précédente

```
preparerPourMac();
```

11 Barème de correction

11.1 Exécution 50%

Si une partie ne fonctionne pas parfaitement, les parties suivantes ne sont pas évaluées. Il ne sert à rien de faire un blitz d'écriture de code qui ne fonctionne pas pour obtenir des points. Assurez-vous de rendre du code qui fonctionne.

11.2 Qualité du code 50%

- Erreurs pénalisantes concernant le respect des normes :
  1. Identificateurs non significatifs (variables, constantes, sous-programmes…).
  2. Aération et/ou indentation et/ou impression laissent à désirer (**80** colonnes max).
  3. Découpage en sous-programmes insuffisant.
  4. Répétition inutile de code dû au manque de sous programmes.
  5. Non utilisation d'un sous-programme lorsque c'est possible ou exigé.
  6. Répétition inutile de code dû à l'incompréhension de l'utilité du paramétrage.
  7. Constantes non définies.
  8. Constantes non utilisées lorsque possible (même dans les commentaires).
  9. Non respect du style Java.
  10. Commentaire d'en-tête des méthodes manquant (description, auteurs, version).
  11. Commentaires des constantes manquants.
  12. Commentaires des variables manquants.
  13. Commentaires non judicieux ou inutile.
  14. Commentaires manquants sur la stratégie employée dans une méthode dont l'algorithme n'est pas évident et nécessite réflexion.
  15. Commentaires mal disposés.
  16. Code inutile.
  17. Méthode qui fait plus d'une tâche.
  18. Affichage dans une méthode de calcul.
  19. Qualité du français dans les commentaires.
  20. Non-respect des droits d'auteurs.
  21. Non utilisation de boucle lorsque possible.
  22. Autres (s'il y a une pratique non énumérée qui ne fait pas de sens).
  23. etc.

Référez-vous aussi aux documents "**ConventionsStyleJavaPourINF111.pdf**" et "**CriteresGenerauxDeCorrectionDuCodeJava.pdf**" fournis avec l'énoncé de ce TP.