README – Compilation 2012

Notre projet a été construit autour de l'API Java Sound. Afin de simplifier son utilisation, nous avons créé une API qui permet de définir des sons et de les jouer. Nous nous proposons ici d'expliquer rapidement comment nous l'avons conçu et de montrer comment s'en servir.

Nous avons créé une classe abstraite `AbstractSound` qui définit le comportement général d'un son dans notre projet. Elle fournit 3 méthodes abstraites : `playSound()`, `stopSound()` et `getSoundDuration()` ; et 2 méthodes concrètes statiques : `setNoteDuration(int duration)` et `setSoundVelocity(int velocity)` qui mettent à jour les 2 champs statiques correspondants.

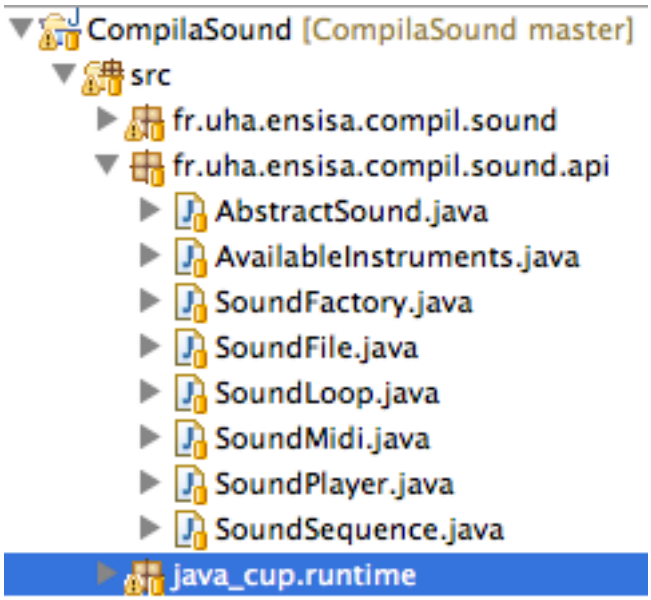

Figure 1 : architecture de l'API du projet

Nous avons ensuite créé 2 implémentations différentes de cette classe qui constituerons nos sons de base : les `SoundMidi`, qui permettent de jouer consécutivement une ou plusieurs notes d'un instrument à définir, et les `SoundFile`, qui se chargent de jouer un fichier son.

Nous avons aussi décidé de créer des structures capables de composer différents sons. Afin que ces compositions restent des sons, les structures dérivent de la classe `AbstractSound`. Nous avons donc des `SoundSequence` qui permettent de jouer consécutivement une liste de sons, et des `SoundLoop`, qui permettent de répéter consécutivement une liste de sons.

Comme tous ces éléments sont des sons, nous pouvons les composer indifféremment et indéfiniment.

Afin de faciliter l'instanciation des différents objets nécessaires depuis le `.cup`, nous avons créé une `SoundFactory` qui permet d'instancier les bons objets. De plus, nous avons créé un `SoundPlayer` qui permet de jouer un son, de jouer simultanément une liste de sons, et de régler la durée d'une note (MIDI) (cela permet de changer le tempo).

La classe `AvailableInstruments` dispose d'une méthode statique `main` exécutable qui affiche sur la sortie standard le numéro et le nom de tous les instruments MIDI disponibles avec la JVM courante.

Dans le package `fr.uha.ensisa.compil.sound` se trouve une classe `Main` qui définit une méthode statique `main` exécutable qui lance le projet de compilation en tant que tel. Par défaut, l'application essaie de parser le fichier `test.txt` présent à la racine du projet. Cette méthode accepte un argument qui n'est autre que le chemin d'un fichier à parser.

Le fichier `test.txt` à la racine du projet montre comment créer des sons midi avec un instrument et une (des) note(s). Le fichier `test2.txt` à la racine du projet montre comment charger des fichiers son.

Le format des notes suit la notation anglaise avec des notes de A à G. Toutes les autres lettres (H à Z) créeront des silences.

Liste des commandes :

- Création d'une note : `load` "nom_instrument" `with_notes` note1; note2; note3; … `end;`
  Renvoie un objet `SoundMidi` capable de jouer les notes demandées avec l'instrument spécifié.
- Import d'un fichier son : `import` "fichier son"`;`
  Renvoie un objet `SoundFile` capable de jouer le fichier son spécifié.
- Création d'une séquence : `sequence` son1; son2; son3; … `end;`
  Renvoie un objet `SoundSequence` capable de jouer la liste de sons spécifiés.
- Création d'une boucle: `repeat` X `times` son1; son2; son3; … `end;`
  Renvoie un objet `SoundLoop` capable de jouer X fois la liste de sons spécifiés.
- Régler la durée des notes (tempo) : `note_duration` XXXX`;`
  Renvoie `vide`, XXXX en millisecondes.
- Joue un son : `play` son`;`
  Renvoie `vide`, joue n'importe quel type de sons.
- Joue une liste de sons simultanément : `play_simult` son1; son 2; son 3; `end;`
  Renvoie `vide`, joue simultanément n'importe quel type de sons.

Des variables peuvent être créées avec l'ensemble des commandes qui retournent une valeur `AbstractSound`. La syntaxe est la suivante :

```
variable = commande params;
```

David BRUN, Benoît SACCOMANO