CHEVALLIER Raphaël M2 Info G.L.
DOUCHEMENT Marc A.O.C.

Rapport Projet-TP Métronome A.O.C.

I. Introduction

Le mini projet-TP mené en A.O.C consiste en la réalisation d'un métronome simple permettant de marquer le tempo par l'intermédiaire de flash et de cliques sonores. L'interface fournie à l'utilisateur permet d'accéder à quatre boutons de commandes (marche, arrêt, incrémenter les temps, décrémenter les temps) ainsi qu'à une molette permettant de régler la vitesse du tempo. Le métronome comporte également un affichage de la valeur du tempo, ainsi que deux diodes permettant de marquer les temps et les mesures.

Deux versions différentes de l'implémentation de l'IHM ont été développées : La première est une implémentation en Swing et la deuxième est un adaptateur permettant de communiquer avec un matériel externe (pour ce TP, un simulateur en java). Le mécanisme permettant de faire fonctionner le métronome est équivalent dans les deux versions. Ainsi, on retrouve un moteur qui permet de gérer l'état du métronome et d'envoyer périodiquement des commandes pour marquer les temps, une interface IHM qui définit l'affichage et le contrôle de l'utilisateur et un contrôleur permettant de contrôler les communications entre le moteur et l'affichage (et inversement). Notre application utilise également les design patterns *command*, *observer*, *factory*, etc. afin de gérer les échanges entre les différents modules.

Dans un premier temps, nous allons détailler la conception commune aux deux versions du métronome, puis, dans un second temps, nous détaillerons les modifications apportées à l'application pour permettre la mise en place d'un adaptateur.

II. Première version – IHM Swing

1 Moteur

L'interface Moteur permet de définir l'état du métronome lors de l'exécution du programme. Il comporte plusieurs attributs modifiables comme la vitesse du tempo *`int tempo`*, le nombre de temps par mesure *`int steps`* ainsi qu'un état de fonctionnement *`boolean started`*. Le moteur du métronome reçoit également une commande de type `CommandMotor` qui permet de marquer les temps. Cette commande est exécutée périodiquement quand le moteur est en état de marche grâce à une `Horloge`. Enfin, cette interface `Motor` étend aussi l'interface `SubjectMotorTempo` lui permettant d'être observée en cas de changement du tempo.

2 IHM

Cette interface doit permettre de représenter graphiquement le métronome. Elle dispose de quatre méthodes (`void flashStep()`, `void clicEmitterStep()`, `void flashMeasure()` et `void clicEmitterMeasure()`) permettant de faire flasher une diode et d'émettre un son afin de marquer les temps et les mesures. Elle comporte également deux attributs : `float wheelPosition` permettant d'accéder à la position courante de la molette et `int displayTempo` permettant de changer la valeur affichée sur l'écran d'affichage du tempo. D'autre part,

l'implémentation de l'IHM reçoit des commandes de type `CommandAdaptor` à exécuter lors des cliques sur ses boutons. Enfin, l'interface IHM étend également l'interface `SubjectWheel` afin de permettre l'envoi de notifications lors du changement d'état de la molette.

3 Contrôleur

Cette interface permet de contrôler et de transmettre des informations entre les deux composants IHM et Moteur. Elle dispose de quatre méthodes permettant de changer l'état du moteur : `void start()` permet de démarrer le moteur si ce dernier est arrêté, `void stop()` permet d'arrêter le moteur si ce dernier est en marche, `void inc()` permet d'incrémenter le nombre de temps par mesure si cette valeur n'a pas déjà atteint son maximum et pour finir, la méthode `void dec()` permet de décrémenter le nombre de temps par mesures si cette valeur n'a pas déjà atteint son minimum. Le contrôleur dispose également d'une méthode `void displayStep()` permettant d'émettre un son et de faire flasher la diode des temps ou des mesures en fonction de l'état du métronome et du temps à marquer. Cette interface étend d'autres parts, les interfaces `ObserverMotorTempo` et `ObserverWheel` afin de transformer le contrôleur en observateur et lui permettre ainsi de recevoir des notifications lors du changement d'état de la molette ou d'un changement de tempo dans le Moteur.

4 Patterns

- **Commandes de l'IHM : Start, Stop, Inc, Dec**

Ces quatre commandes, permettant d'exécuter les actions lors de cliques sur les boutons de l'IHM, implémentent toutes l'interface `CommandAdaptor`. Elles sont créées et transmises à l'IHM grâce à la fabrique de création (Pattern Factory) `Metronome`. Elles sont invoquées par l'IHM (lors des événements `mousePressed` pour la première version du métronome) et appellent des méthodes de la classe `Controller`.

- **Commandes du Moteur : Flasher**

La commande `CmdDisplayStepImpl` implémente l'interface `CommandMotor` et permet d'exécuter une méthode sur le `Controller` afin de marquer les temps et les mesures. Cette commande est créée et transmise au `Moteur` grâce à la fabrique `Metronome` et est invoquée périodiquement par l'horloge du Moteur.

- **Observeur sur le tempo du Moteur**

Afin de notifier les changements de tempo du `Moteur`, une implémentation du pattern observer a été mise en place entre le `Moteur` et le `Controller`. Deux interfaces ont donc été créées : `SubjectMotorTempo` est le sujet de l'observer implémenté par le `Moteur` et `ObserverMotorTempo` représente l'observer implémenté par le `Controller`. La méthode updateObservers() du sujet est appelée lors de chaque exécution de la méthode setTempo(int tempo) du Moteur. Elle permet de notifier au `Controller` un changement d'état en appelant la méthode `updateDisplayTempo(SubjectMotorTempo s)` de l'observer.

- **Observeur sur la molette de l'IHM**

Afin de notifier les changements de valeurs de la molette de l'`IHM`, une implémentation du pattern observer a également été mise en place entre le l'`IHM` et le `Controller`. Deux interfaces ont

été créées : `SubjectWheel` est le sujet de l'observer implémenté par le `IHM` et `ObserverWheel` représente l'observer implémenté par le `Controller`. La méthode `updateObservers()` du sujet est appelée lors de la détection par l'IHM d'un changement de valeur de la molette (dans la première version, c'est un `ChangeListener` qui permet de contrôler ces changements et d'envoyer une notification quand la valeur est stabilisée). Cette notification est représentée par l'appel de la méthode `updateWheel(SubjectWheel s)` de l'observer.

III. Deuxième version – IHM Hardware avec adaptateur

1 Simulateur de matériel

Pour cette deuxième version du Métronome, un simulateur de matériel a été créé afin de pouvoir représenter les communications de l'application avec un matériel externe. Ce simulateur comporte les mêmes spécifications et les mêmes interfaces que le vrai matériel, seule son implémentation a été faite en Java.

2 Pattern adaptateur

Afin de permettre au Métronome de pouvoir communiquer avec une interface d'un matériel externe (dans cette deuxième version, un simulateur en Java), aucune modification n'a été apporté à la logique de fonctionnement de l'application. Ainsi, on retrouve les mêmes interfaces et les mêmes implémentations du `Motor` et du `Controller`, seule une nouvelle implémentation de l'`IHM` a été mise en place. Cette nouvelle implémentation correspond en fait à un `ConcreteAdapter` du design pattern `Adapter` et va permettre d'adapter les communications entre l'`IHM` et l'interface du matériel. Ainsi, une méthode `readHardware()` a été créée afin de vérifier l'état du matériel. Elle permet notamment de sauvegarder l'état courant afin de détecter les changements dans l'interface utilisateur (appui sur un bouton ou changement de valeur de la molette). D'autre part, cette nouvelle implémentation de l'IHM permet toujours d'accéder à la valeur de la molette, de changer la valeur d'affichage du tempo et de déclencher l'exécution des méthodes permettant de faire flasher les diodes ainsi que d'émettre un son afin de marquer les temps et les mesures.

3 Pattern commande ReadHardware

Une commande permettant d'appeler la méthode `readHardware()` a également été créée. Elle a l'Adapter à la foi comme client et comme receiver, mais elle est invoquée par une Horloge créée par l'Adapter afin d'être exécutée périodiquement.

IV. Tests

Pour vérifier le bon fonctionnement du métronome, des tests unitaires et d'intégration ont été réalisés. Ces tests sont les mêmes entre la v1 à v2.

1 Unitaires (motor)

Un *mock* du contrôleur à dû être créé afin de tester unitairement le moteur. Ces tests sont détaillés dans la classe test `TestMotorMetronomeIU` du package test (motor).

2 Intégration

Ces tests permet de vérifier la cohérence entre l'état de l'IHM Swing et l'état du moteur du métronome (dans les deux sens). Ils sont détaillés dans la classe `TestIHMImplUI` du package test (ihm).

V. Conclusion

Nous pouvons voir que notre application respecte le cahier des charges. Ce projet nous a permis de mieux apprendre à utiliser les partons de conception impliquant une diminution du couplage entre les classes principales.