Stéphane TOYO DEMANOU 18/04/2021

Amir CHERKAOUI

LOG2410 TP5 : Conception à base de patrons

2. Identifier des patrons de conception utiles pour votre cas d'étude

- Patron État:

L'intelligence artificielle de notre application PolyPiano doit accomplir de multiples tâches, toutes différentes les unes des autres (afficher la partition musicale, enregistrer et analyser la performance de l'élève, ect…). De plus, il ne faut pas oublier que l'application aura différentes fonctionnalités selon si le profil utilisateur est celui d'un élève ou celui d'un professeur. C'est ici que le patron État rentre en scène, il va nous permettre de modifier le comportement de la classe principale gérant l'application afin de pouvoir remplir la fonctionnalité dont l'utilisateur aura besoin à un moment donné. Cette approche va nous éviter les blocs massifs d'instances conditionnelles (if-else, switch-case) qui peuvent vite devenir une faiblesse grave au fur et à mesure de l'évolution du projet.

- Patron Singleton

L'utilité principale de PolyPiano est l'analyse de la performance musicale des étudiants. Pour ce faire, parmi tout ce que propose PolyPiano, il y a un programme chargé d'exécuter cette tâche. Ce programme est unique, qui que soit l'étudiant, il va opérer de la même façon. Le fait qu'il soit en un seul exemplaire amène l'idée d'utiliser le patron Singleton. Ce patron garantit une seule instance d'une classe, et fournit un point d'accès global a celle-ci, ce qui convient parfaitement à notre situation. On peut alors créer une classe qui contiendra la méthode analysant la performance de l'étudiant.

3. Discussion des avantages et des inconvénients

- Patron État :

Avantages :

- Dans notre contexte, les différents états codés représentant les fonctionnalités de notre application seront chacun séparés dans des classes distinctes, ce qui assurera le respect du principe de responsabilité unique. Ainsi, une fonctionnalité n'entrera pas en conflit avec une autre.
- Il sera plus simple d'ajouter de nouveaux états; et donc classes; sans avoir à modifier les états/classes existantes. Cela améliore grandement la maintenabilité de notre application PolyPiano ainsi que son potentiel évolutif puisqu'il y sera plus facile d'y ajouter des fonctionnalités. C'est le principe ouvert/fermé.

Inconvénients :

- Ce patron peut être excessif lorsque l'on n'a pas beaucoup d'états ou de transitions à implémenter. Si l'application PolyPiano finit par contenir moins de fonctionnalités que prévu, on se sera compliqué la tâche pour rien.

- Patron Singleton :

Avantages :

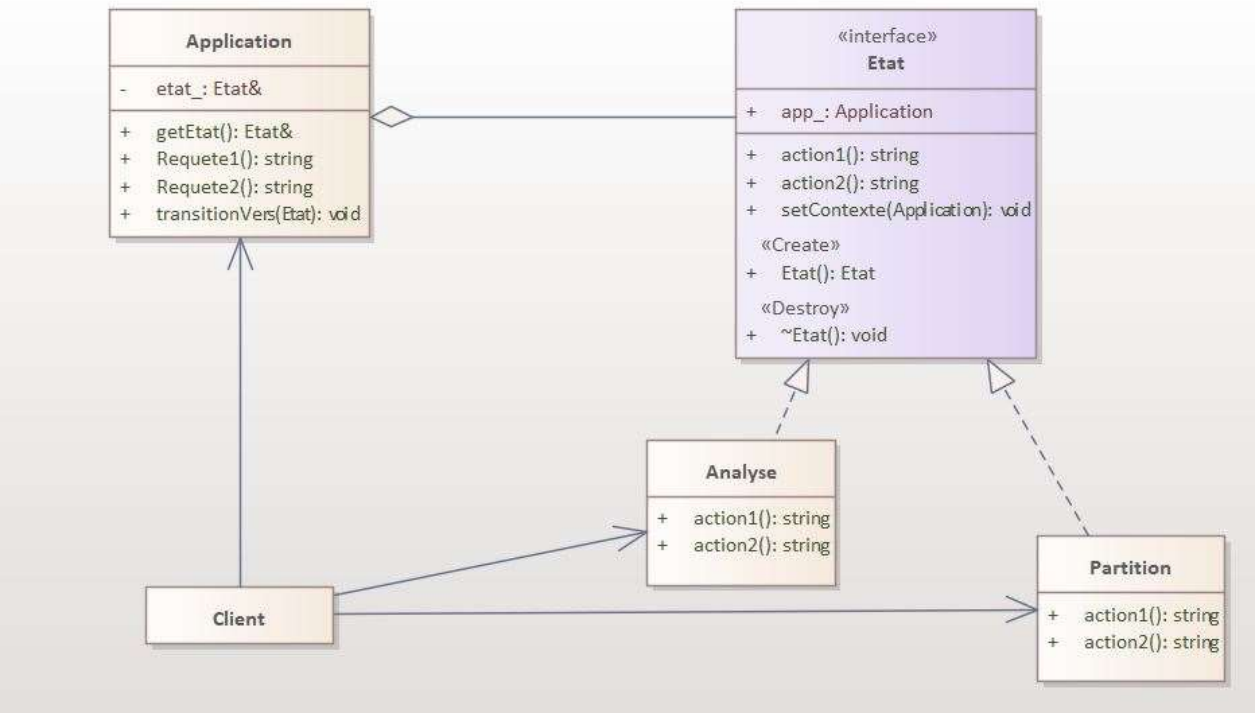
- La classe créée est unique, et on a un point d'accès global à celle-ci, toute instance d'étudiant va faire appel au même objet pour analyser la performance.
- La classe est initialisée une seule fois tout le long, ce qui est très efficace et excellent au niveau de la performance de l'entièreté de PolyPiano.
- Le fait qu'il n'y ait qu'une instance faciliterait énormément le débogage lors de l'implémentation de PolyPiano.

Inconvénients :

- Le patron Singleton viole le principe de responsabilité unique, stipulant que chaque classe doit avoir une responsabilité distincte.
- Dans notre cas, les tests unitaires pourraient s'avérer difficiles, il est difficile d'isoler une classe dépendant d'une classe Singleton, donc certains tests unitaires pourraient s'avérer plus compliques que d'autres.
- S'il y a une mauvaise conception dans notre implémentation, le patron Singleton peut masquer cela, et causer des problèmes plus tard.

4. Les diagrammes de classes détaillés

- Le Patron État :



- Le patron Singleton :