POLYTECHNIQUE
MONTRÉAL

LOG1410
Analyse et conception de logiciels

Laboratoire 5

Soumis par:
Ioana Daria Danciu - 2081744
Alexandre Gélinas - 2083465

Le 18 avril 2020

Patrons de conceptions utiles pour notre cas d'étude

Observateur : interface

Le patron Observateur serait pratique pour la gestion de l'interaction de l'interface utilisation avec l'utilisateur. Dès que l'utilisateur appuie sur un bouton, il faudra que le système réagisse à ce changement. De ce fait, lorsque l'enseignant voudrait ajouter les droits d'auteurs à la partition, le système devrait capter ce que l'enseignant fait pour bien enregistrer ce que celui-ci veut obtenir.

Fabrique : modification des symboles

Le patron Fabrique serait utile pour la modification des symboles. En effet, nous pourrions créer une classe mère déclarant tous les symboles pouvant se trouver sur une partition de musique et ensuite modifier leurs aspects, comme leur couleur, dans des classes dérivées. On pourrait également ajouter de nouveaux symboles de cette manière.

Singleton : serveur infonuagique

Le patron Singleton serait très pratique en vue de la sauvegarde de différents fichiers. Comme il y a qu'une seule base de donnée, soit le serveur ou la mémoire interne par exemple, on ne doit avoir qu'une seule instance de ceux-ci pour bien représenter le système. De plus, le patron pourrait être utilisé pour les événements comme pour la souris ou le clavier dans l'application.

Commande : menu

Le patron Commande serait utilisé pour la gestion de menu. Avec les boutons du clavier ou de l'application, nous pourrons ainsi revenir en arrière d'une action et exécuter une commande faite par l'utilisateur. Ainsi, si l'utilisateur appuie sur le bouton "Retour" de l'application, l'application va revenir au menu précédent ou va revenir à la position avant la dernière action effectuée.

Discussion des avantages et des inconvénients

Observateur :

Le patron observateur apporte beaucoup d'avantages pour l'interface d'une application. Il permet de facilement faire circuler l'information entre plusieurs classes pour bien répondre aux actions de l'utilisateur. De plus, il simplifie la gestion des événements car ceux-ci peuvent être notifiés aux classes qui observent les classes qui sont observées. De plus, il sera très simple d'ajouter des classes qui observent et qui sont observées car nous devons simplement les faire hériter des classes de base, soit les classes gérant le patron observateur. Cependant, il est plus difficile d'exécuter une liste de tâches dans un ordre précis car seul l'observateur pourra savoir l'ordre des classes qui seront notifiées. Il pourrait donc être bien de se créer une liste avec une priorité de notification pour avoir un ordre plus simple de ce qu'on veut qui soit notifié en premier.

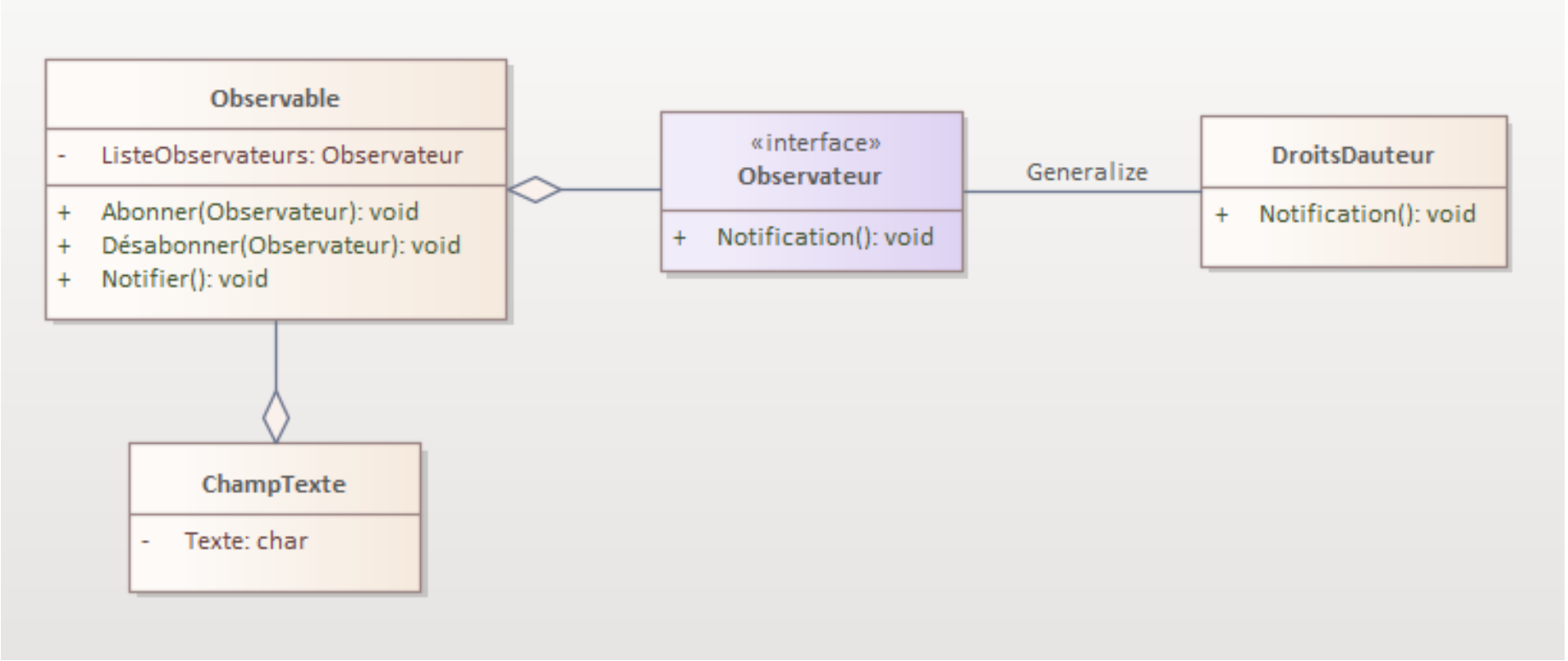
Fabrique :

Il existe plusieurs avantages à l'utilisation du patron Fabrique lors de la gestion des symboles de musique. En effet, il permet de réduire le couplage entre la classe de base et la façon dont les objets sont construits, ce qui facilite l'ajout de nouveaux symboles. Pour ce faire, il est possible de créer une classe dérivée, ou d'en modifier une déjà existante, si l'ancien type qu'elle définissait n'est plus nécessaire. Cela empêcherait la complexification des tests à faire lors de la création d'un nouveau type de symbole. On éviterait alors un enchaînement de structures conditionnelles et donc, la vérification du type de l'objet à créer puis l'appel de l'une des méthodes de création à chaque fois qu'une nouvelle instance est créée. Il devient plus simple d'avoir plusieurs méthodes dérivées capables de construire qu'un seul type d'objet. Toutefois, il existe des désavantages: nous devons construire une nouvelle classe dérivée à chaque fois que l'on souhaite ajouter un nouveau symbole. Cela peut rendre la manipulation, donc la création des objets, plus difficile à comprendre. La mémoire doit également contenir plus de données. On pourra donc créer plusieurs fabriques dans le cas où l'application se développe, voire même d'utiliser le patron de Fabrique abstraite.

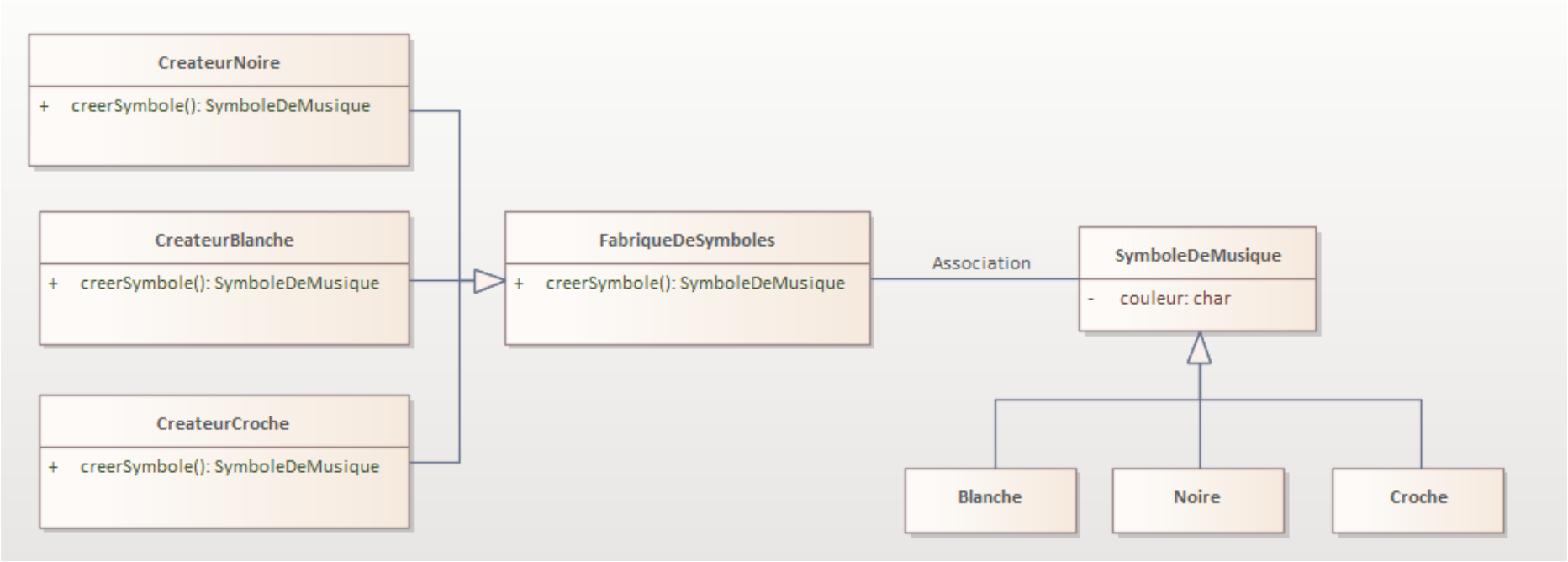
Singleton : Les grands avantages du patron singleton est son accès global. On pourra avoir accès à la base de données, par exemple, à l'intérieur de tout le programme. De plus, nous n'aurions pas à savoir si l'objet est déjà créé ou non car le premier appel va toujours créer l'instance désirée. Cependant, il pourrait être plus difficile de penser à enregistrer nos informations à plusieurs endroits car la classe est unique. On pourrait donc créer une classe qui s'occupe du stockage sur différents emplacements pour régler ce problème. De plus, il pourrait être difficile de travailler avec plusieurs tâches de travail pour le programme si celui-ci se développe énormément étant donné que chaque tâche doit avoir la même instance. Il serait donc recommandé de travailler avec un programme qui ne fait qu'une tâche à la fois.

Commande : Le patron commande facilite la gestion de la sauvegarde d'une suite d'événements qui pourrait avoir lieu. De ce fait, il pourrait être réutilisé dans plusieurs autres domaines pour l'avenir de l'application comme un historique des partitions déjà jouées ou un historique des cours suivis ou à venir. De plus, il permet de facilement annuler une tâche avant même que celle-ci ne soit exécutée. Les désavantages de ce patron sont surtout pour la conception du code. Chaque classe devra implémenter la méthode nécessaire à la classe qui commande pour pouvoir l'exécuter. C'est une des étapes nécessaires pour que le patron puisse bien fonctionner.

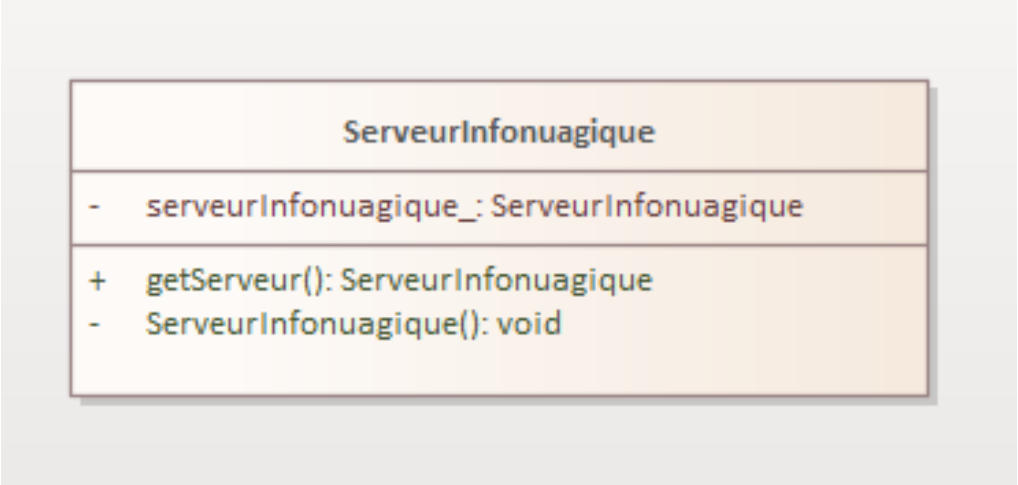
Patron Observateur : Interface de l'application



Patron Fabrique : Symboles de musique



Patron Singleton : Serveur infonuagique



Patron Commande : Bouton de commande