UNIVERSITÉ DE BORDEAUX

MASTER 1 INFORMATIQUE

ARCHITECTURE LOGICIELLE

Projet Micro éditeur de figures géométriques

Auteurs
Fabien ALAZET
Nicolas DEGUILLAUME

24 Mai 2020

Table des matières

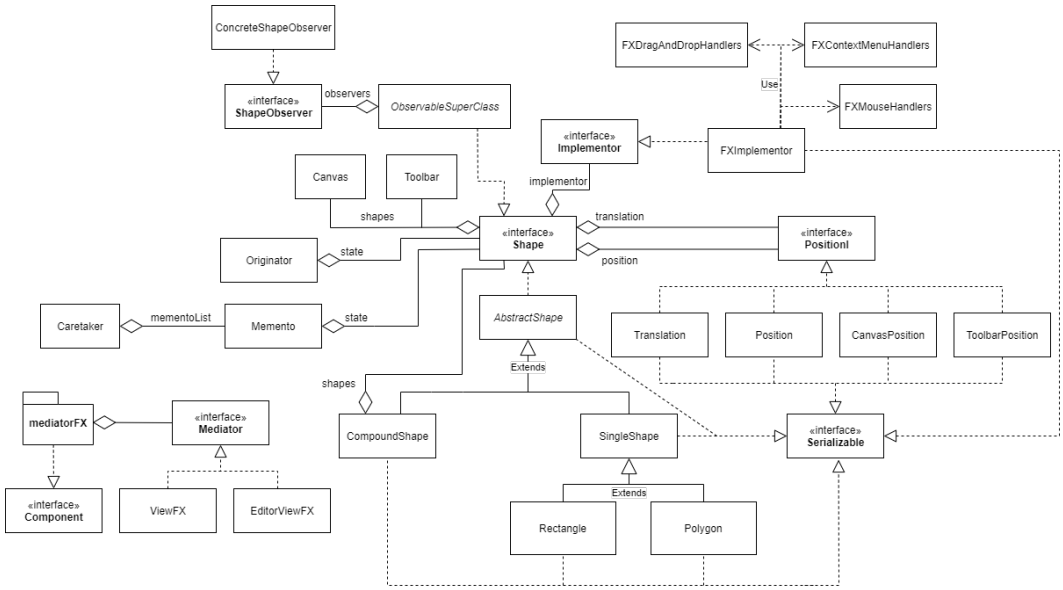
Introduction

Comme projet d'Architecture Logicielle, nous avons eu à créer un logiciel de dessin vectoriel dont le principe est de créer de nouveaux dessins en utilisant/modifiant/combinant des dessins existants. Plusieurs fonctionnalités sont proposées, qui sont les suivantes :

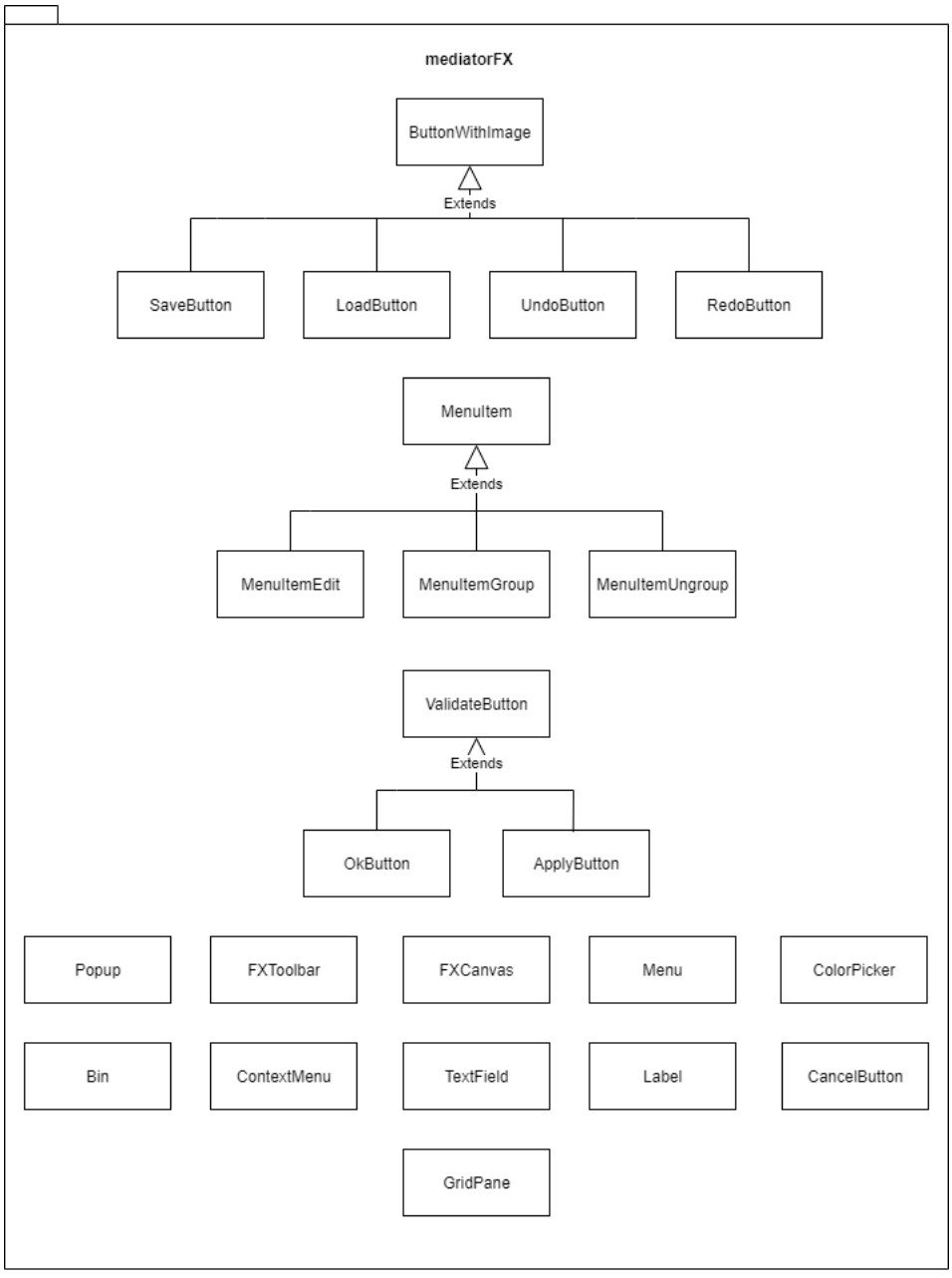
— Proposer au minimum 2 objets : le polygone régulier et le rectangle
— Sélectionner un objet depuis un menu graphique (toolbar), et le positionner sur notre dessin – (whiteboard) en utilisant le glisser-déposer (drag and drop)
— Créer des groupes d'objets et sous-groupes d'objets, sous-sous-groupes etc
— Dissocier un groupe d'objet
— Modifier la taille, position, etc… de nos objets ou groupes d'objets une fois ceux-ci incorporés dans le dessin
— Ajouter des groupes d'objets ou des objets paramétrés à notre toolbar en les déposants sur la toolbar (drag and drop)
— Annuler ou refaire une opération
— Sauvegarder un document et charger un document
— Sauvegarder l'état du logiciel (toolbar) et le recharger au démarrage.

Nous avons utilisé certains des Patrons de Conception vus en cours que nous avons jugés pertinents dans la réalisation des fonctionnalités demandées ainsi que dans l'optique de ne pas assujettir notre code à un moteur graphique (ici, javaFX) et pour permettre la réutilisation de code.

UML du projet



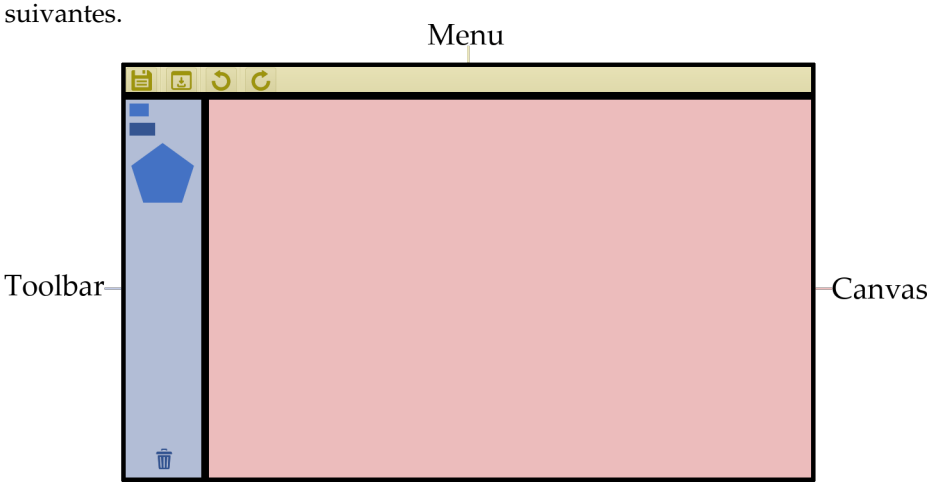
Contenu du package mediatorFX :



Design Patterns Utilisés

1 Remarque

Notre application peut se découper en 3 zones principales : le menu, le canevas (canvas) et la barre d'outil (toolbar) voici un schéma pour vous aider à les situer sur notre application étant donné que l'on va en parler dans les parties suivantes.



2 Pattern Prototype

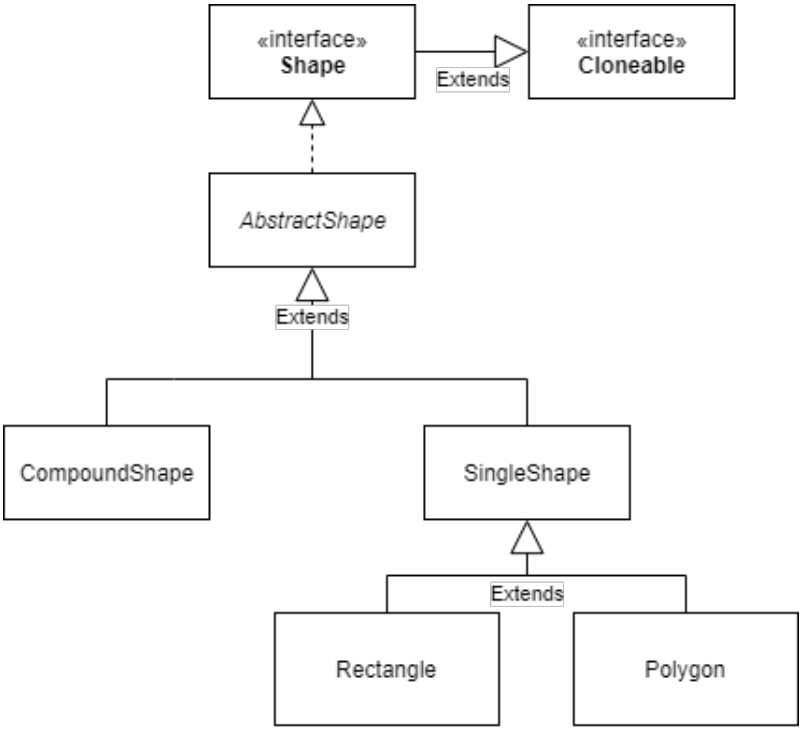
2.1 Intention

Le pattern Prototype permet de copier des objets existants sans rendre le code existant dépendant de leur classe. L'utilisation du pattern permet de cloner des objets sans coupler à leur classe concrète. Elle permet de se débarrasser du code d'initialisation répété en faveur du clonage de prototypes pré-construits.

2.2 Problématique

Nous voulons créer une logiciel de dessin qui fonctionne en déplaçant des formes de la barre d'outils vers le canevas. Une fois placées, nous voulons changer les attributs des formes. Comment faire pour ne pas modifier le code de la barre d'outils à chaque fois que l'on veut y ajouter une forme.

2.3 Implémentation



3 Pattern Singleton

3.1 Intention

Le pattern Singleton est un design pattern de création qui permet d'assurer qu'une classe n'a qu'une instance, tout en fournissant un point d'accès global à cette instance.

3.2 Problématique

Nous voulons créer un logiciel de dessin qui se compose d'un canevas et d'une barre d'outils uniques, qui fonctionne en déplaçant des formes géométriques de la barre d'outils au canevas et inversement. Il faut donc faire une classe pour la barre d'outils et une pour le canevas. Ces deux classes ne doivent pas pouvoir être instanciées plus d'une fois, et doivent être accessible depuis n'importe quelle autre classe car elles sont essentielles.

3.3 Implémentation

Nos classes Singleton sont les classes : view.ViewFX.java, view.EditorViewFX.java, model.Caretaker.java, model.Canvas.java, model.Toolbar.java, model.FXImplementor.java, view.Originator. Pour chacune d'entres elles nous avons :

- Ajouté un champ `Instance` pour stocker l'instance du singleton
- Déclaré une méthode `getInstance()` qui crée objet au premier appel et qui ensuite renvoie `Instance`
- Mis la visibilité des contructeurs à privée

4 Pattern Composite

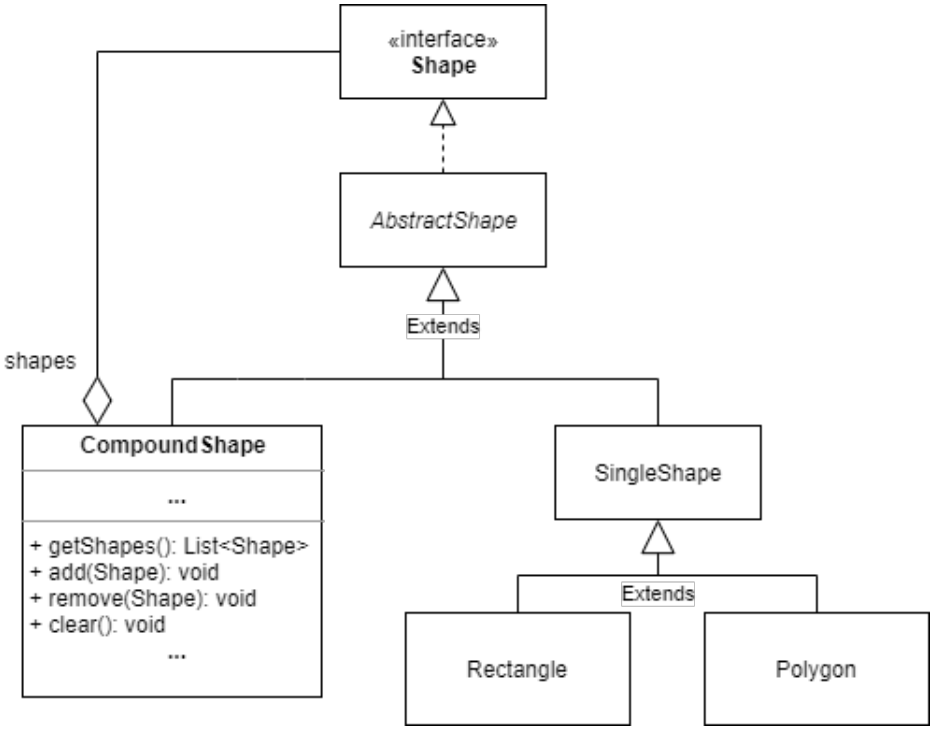
4.1 Intention

Le pattern Composite est un pattern structurel qui permet de composer des objets en structures, puis de travailler avec ces structures comme si elles étaient des objets individuels. Grâce à ce pattern, il est plus facile de travailler avec des arborescences complexes.

4.2 Problématique

Nous voulons créer un logiciel qui permet de créer des formes géométriques, et de les grouper ensemble pour former un groupe de formes géométriques. Il est ensuite possible d'éditer ce groupe de formes géométriques. Comment faire pour avoir une architecture permettant de stocker des formes géométriques, des groupes de formes géométriques.

4.3 Implémentation



5 Pattern Bridge

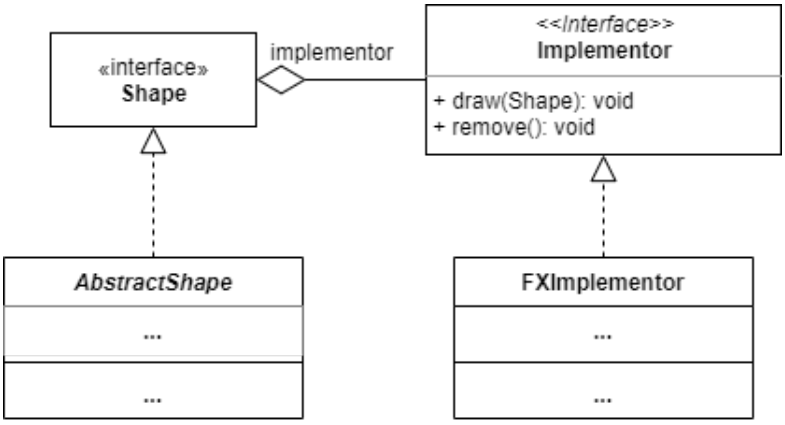
5.1 Intention

Le pattern Bridge est un pattern structurel qui permet de découpler une abstraction de son implémentation afin que ces éléments puissent être modifiés indépendamment l'un de l'autre.

5.2 Problématique

Nous avons une application qui fonctionne avec JavaFX, et nous aimerions qu'elle puisse fonctionner avec Swing sans avoir à modifier le code existant.

5.3 Implémentation



6 Pattern Mediator

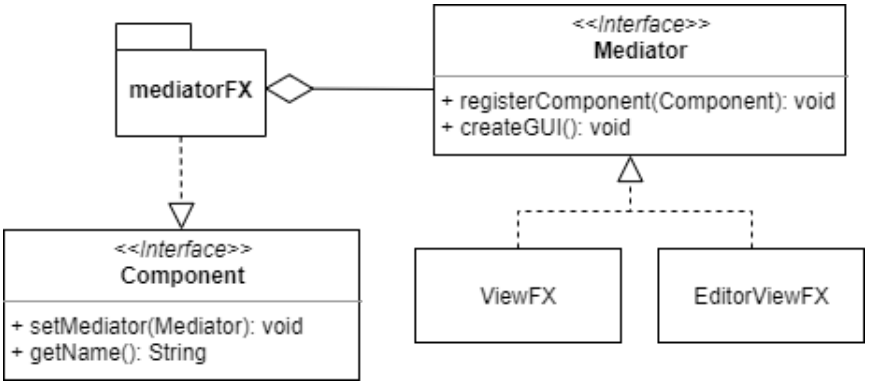
6.1 Intention

Le pattern Mediator est un pattern de comportement qui permet de réduire des dépendances chaotiques entre des objets. Ce pattern limite les interactions directes entre ces objets et les force à collaborer via un objet qui sert de médiateur.

6.2 Problématique

Nous utilisons JavaFX pour réaliser l'affichage de notre application. Pour cela, nous utilisons plusieurs composants tels que des boutons, des labels, des champs de texte, etc. Nous avons plusieurs fenêtre d'éditeur en fonction de ce qui est sélectionné à l'écran, et nous faisons apparaître différents champs sur ces fenêtres. Nous voulons faire en sorte que certains objets interagissent avec différents éléments en fonction de la fenêtre affichée. Nous avons donc besoin d'une classe qui permet de gérer ces interactions.

6.3 Implémentation



7 Pattern Memento

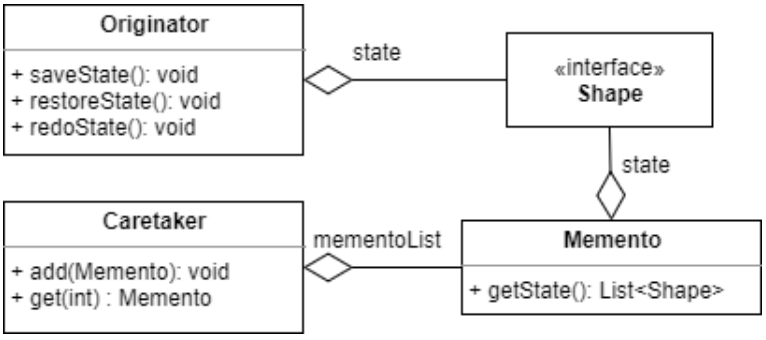
7.1 Intention

Le pattern Memento est un pattern de comportement qui permet de sauvegarder et de restaurer un état précédent d'un objet, sans réléver les détails de son implémentation.

7.2 Problématique

Nous voulons instaurer un principe d'undo/redo dans notre application, pour que l'utilisateur puisse annuler les actions qu'il a effectuée une par une pour revenir à l'état dans lequel était le logiciel à son lancement si il faut. Nous voulons aussi qu'il puisse refaire une action annulée.

7.3 Implémentation



8 Pattern Observer

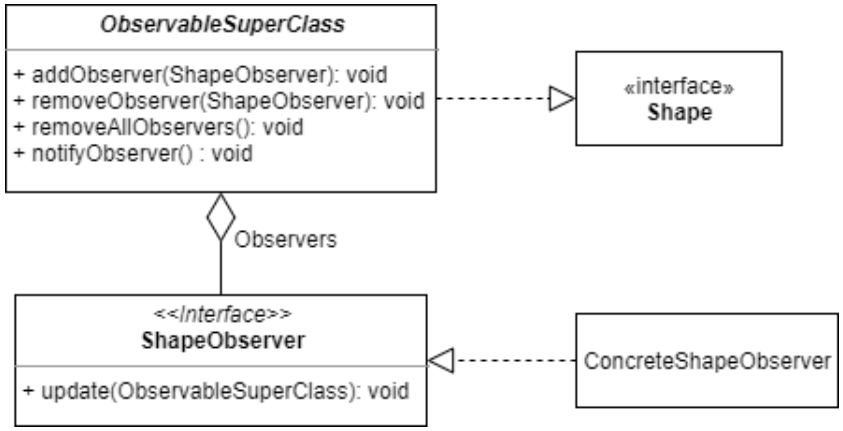
8.1 Intention

Le pattern Observer est un pattern de comportement qui permet de définir une interdépendance de type un à plusieurs, de telle façon que quand un objet change d'état, tous ceux qui en dépendent soient notifiés et automatiquement mis à jour. Un observateur peut donc notifier plusieurs objets a propos de n'importe quel évènement qui se produit à un l'objet qu'il observe.

8.2 Problématique

Nous avons une application qui permet d'afficher des formes géométrique, qu'il est possible d'éditer, ou de déplacer. Il faut donc notifier le canevas ou la barre d'outil à chaque changement afin d'afficher les nouvelles formes géométriques modifiées.

8.3 Implémentation



9 Conclusion