Projet Persistence

Auteurs

- Aladenise Arthur
- Wicaksono Pradityo Adi

Description

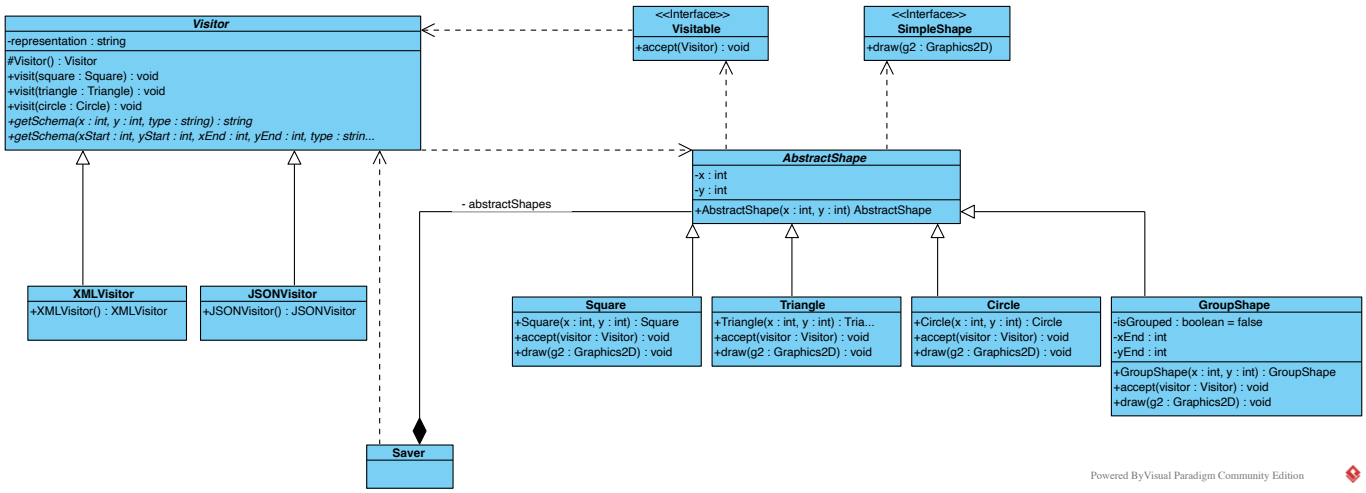
Ce projet est réalisé dans le cadre du cours Patrons et Composants. Le but de ce projet est de mettre en pratique les différents patrons de conception vus en cours, mais aussi la mise en place de test. Ces différentes pratiques sont appliquées à travers un projet d'éditeur graphique.

Vous trouverez le lien vers le dépôt git du projet ici

Patrons de conception

Patron visiteur :

Pour réaliser la persistence des formes à travers des exports en XML et JSON, nous avons utlisé le patron visiteur.



> Tous les attributs et méthodes non nécessaires à la compréhension du patron ont été retirés des diagrammes.

Pour implémenter le patron *Visiteur*, nous avons donc une classe abstraite `Visitor` qui définit les méthodes `visit` pour chaque type de forme. Ainsi que les classes `XMLVisitor` et `JSONVisitor` qui implémentent ces méthodes.

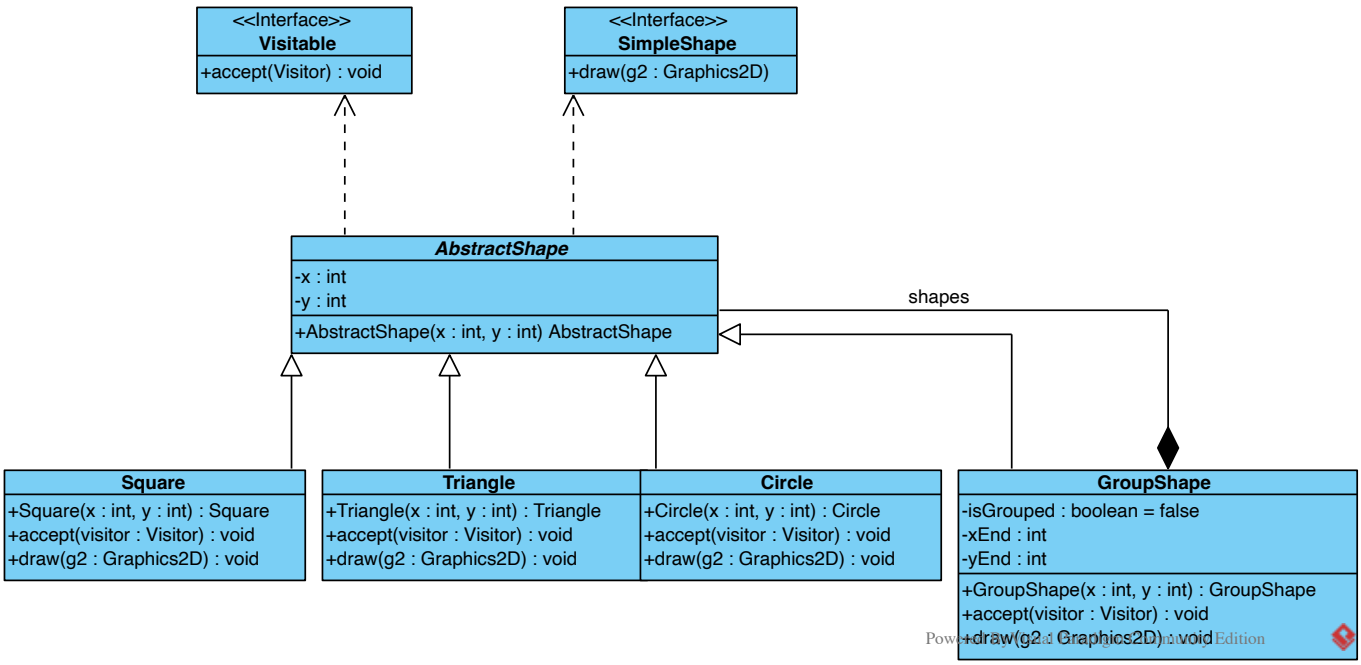
Nous avons ensuite une interface `Visitable` qui définit la méthode `accept` qui permet d'accepter un visiteur. Et l'interface `Shape` qui définit la méthode `draw` pour dessiner la forme. (NB : Cette interface n'est pas liée au patron *Visiteur*.)

Puis, nous avons la classe abstraite `AbstractShape` qui implémente l'interface `Visitable` et `SimpleShape`. Enfin, nous avons les classes `Circle`, `Rectangle`, `Triangle` et `GroupShape` qui héritent de `AbstractShape` et implémentent l'interface `Shape`. Les forment implémentent donc la méthode `accept` qui appelle la méthode `visit` du visiteur passé en paramètre.

Dans notre projet, c'est la classe `Saver` qui fait office de client pour le patron *Visiteur*.

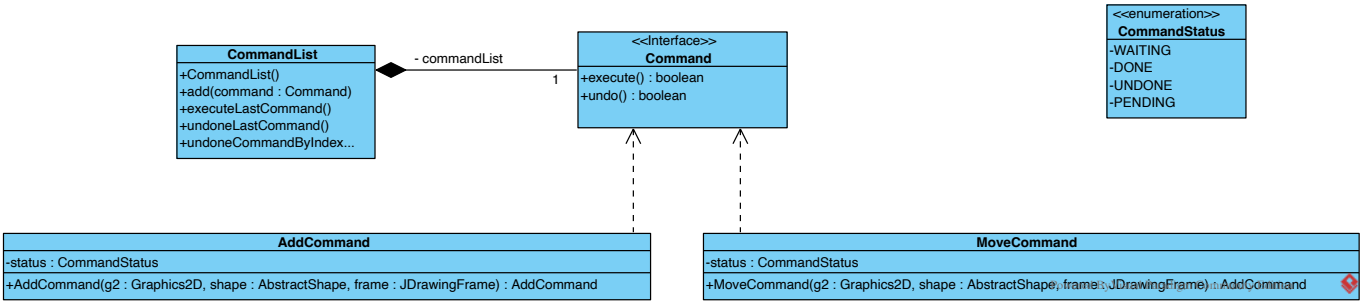
Patron composite :

Pour l'itération *visiteur du midi(3)*, nous devions implémenter le groupement des formes. Pour cela, nous avons implémenté le patron *Composite*.



Nous avons alors ajouté une nouvelle forme de type `GroupShape` qui hérite de `AbstractShape` et qui est composée d'une liste d'`AbstractShape`. Ainsi, `GroupShape` possède une liste de formes qui peuvent être des `Circle`, `Rectangle`, `Triangle` ou `GroupShape`. `GroupShape` est donc le *composite* du patron *Composite* et `AbstractShape` est le *composant*. Les classes `Circle`, `Rectangle` et `Triangle` sont quant à elles les *feuilles* du patron *Composite*.

Patron de commande :



Comme nous utilisons deux commandes, l'ajout et le déplacement d'une forme, nous utilisons le patron de *Commande*. Grâce à celui-ci, nous pouvons facilement implémenter le `undo` car chaque command sait ce quelle a à faire pour etre faite ou défaite.

Dans notre projet, la classe `CommandList` fait office d'*invoker* pour le patron.

État du projet

À ce jour, le projet implémente presque toutes les fonctionnalités du visiteur du midi(3).

- [x] Ajout de formes
- [x] Undo sur l'ajout
- [x] Déplacement de formes
- [x] Undo sur le déplacement
- [x] Groupement de formes
- [x] Undo sur le groupement
- [x] Export en XML
- [x] Export en JSON
- [x] Tests unitaires
- [x] Réintégration du challenge *visiteur 4*
- [ ] Déplacement de groupe

> Lors du undo de groupement, il peut y avoir des problèmes d'affichage des formes, mais un export permet de vérifier que les formes sont toujours présentes