Tableau d'application des patrons de conception

Patron Observer

Élément du patron	Application dans le projet
Subject	`Perspective` et `Image` implémentent l'interface `Subject`
Observer	`VueAbstraite` et ses sous-classes implémentent l'interface `Observer`
Notify	Méthode `notifyObservers()` dans `Perspective` et `Image`
Update	`VueAbstraite.update(Subject)` qui appelle `dessiner()`
Attach	`perspective.attach(this)` dans le constructeur de `VueAbstraite`
Intérêt	Les vues se mettent à jour automatiquement quand la `Perspective` change

Justification : Nous avons voulu mettre à jour les vues quand la perspective est modifié.

Patron Command

Élément du patron	Application dans le projet
Command	Interface `Command` avec méthodes `execute()` et `undo()`
ConcreteCommand	`ZoomCommand`, `TranslationCommand`, `CommandeColler`, `CommandeCopier`
Invoker	`CommandManager` qui exécute les commandes
Client	`ControleZoom`, `ControleTranslation`, `GestionnaireInterface` qui créent les commandes
Receiver	`Perspective`et `PerscpectiveMediator` dont l'état est modifié par les commandes
Intérêt	Permet l'historique d'actions avec undo/redo

Justification : Nous avons voulu que les différents types de commande soient ré-exécuté ou défate, ainsi on peut avoir un historique des actions faites.

Patron Strategy

Élément du patron	Application dans le projet
Strategy	Interface `ControleSouris`
ConcreteStrategy	`ControleZoom`, `ControleTranslation`
Context	`VueInteractive` qui utilise une stratégie
Client	`ImageController` qui configure les stratégies
Intérêt	Change dynamiquement le comportement des vues face aux événements souris

Justification : Comme nous avions deux types de modification avec la souris, nous avons décidé d'implémenter le patron stratégie afin de que les vues aient leur propre comportement avec la souris.

Patron Memento

Élément du patron	Application dans le projet
Memento	Classe `PerspectiveMemento`
Originator	`Perspective` qui crée et utilise des mementos
Caretaker	`Sauvegarde` qui stocke les mementos
Intérêt	Permet de sauvegarder et restaurer l'état des perspectives

Justification : Le patron Memento permet de capturer l'état interne d'un objet sans exposer sa structure, pour ensuite le restaurer.

Patron Singleton

Élément du patron	Application dans le projet
Instance	`CommandManager.instance`et `GestionnaireInterface`
GlobalAccessPoint	`CommandManager.getInstance()` et `GestionnaireInterface.getInstance()`
PrivateConstructor	`private CommandManager()` et `private GestionnaireInterface()`
Intérêt	Garantit une instance unique du gestionnaire de commandes

Justification : Le patron Singleton a été utilisé pour garantir qu'une seule instance de certaines classes critiques soit accessible de manière centralisée à travers l'application.

Patron MVC

Élément du patron	Application dans le projet
Model	`Image` et `Perspective`
View	`VueAbstraite` et ses sous-classes (`VuePrincipale`, `VueSecondaire`, `VueFixe`)
Controller	`ImageController`, `ControleZoom`, `ControleTranslation`
Intérêt	Sépare les données, leur présentation et les interactions utilisateur

Patron Médiateur

Élément du patron	Application dans le projet
Mediator	`Mediator`
Colleague	`CommandeCopier`, `CommandeColler`
ConcreteMediator	`ImageController`, `ControleZoom`, `ControleTranslation`

Élément du patron	Application dans le projet
Intérêt	Gère la communication entre les collègues

Justification : Le patron Mediator simplifie les interactions entre les classes responsables de copier/coller en centralisant les communications.