CHOIX DES DP

Groupe 4

Groupe 4

Membres du groupe

- NWEDJIWE Enzo
- BALOG Phil
- SOUOPGWI Freedy
- NGAMALEU Iris

Design Patterns :

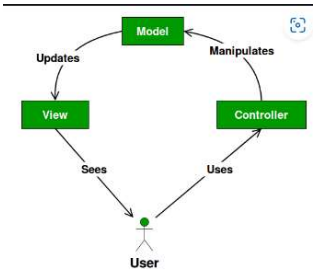
1. Design Pattern **MVC** :

Le Design Pattern **MVC** pour **Modèle-Vue-Contrôleur** fait partie des plus connu et reste particulièrement presqu'incontournable dans le monde de développement logiciel.

Il permet notamment de séparer bien organiser le code source en définissant les fichiers à créer et leurs différents rôles.

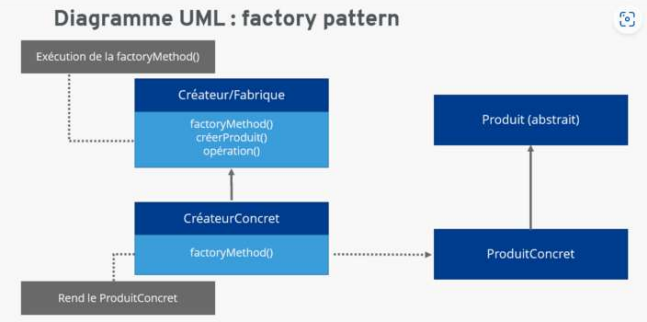
Le but de MVC est justement de séparer la logique du code en trois parties que l'on retrouve dans des fichiers distincts :

- **Vue :** La vue représente la partie de l'application que l'utilisateur a sous ses yeux. Elle est donc implémentée par une IHM. Elle s'occupe des interactions avec l'utilisateur : présentation, saisie et validation des données.
- **Modèle :** Le modèle est la partie de l'application qui exécute la logique métier, c'est-à-dire le code applicatif de l'application. Il la responsabilité de : récupérer les données, et de les manipuler conformément à l'application (le traitement, la validation, l'association et beaucoup d'autres tâches).
- **Contrôleur :** C'est la couche intermédiaire qui permet d'interagir entre les deux (2) autres couches du modèles (Vue et Modèle). Il a pour principale rôle de **Superviser** l'application. Le contrôleur se charge de savoir quelles données doivent être chargées depuis la base de données, la vérification du niveau d'authentification, appeler le **modèle** correspondant et retourner la **vue** adéquate.



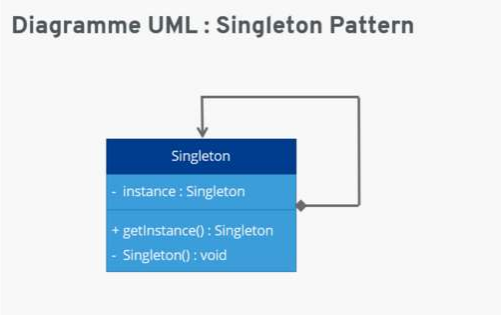
2. Design Pattern **Factory** :

Le design Pattern Factory permet de fournir une classe qui va s'occuper de l'instanciation de toutes les classes du programme. Pour la mise en œuvre, les développeurs ont recours au patron de conception fabrique, également appelé Factory pattern, qui donne son nom au modèle.
Le Factory pattern vise à résoudre un problème fondamental lors de l'instanciation, c'est-à-dire la création d'un objet concret d'une classe, dans la programmation orientée objet : **créer un objet directement au sein de la classe,** qui a besoin de cet objet ou devrait l'utiliser, est possible en principe, mais **très rigide**.



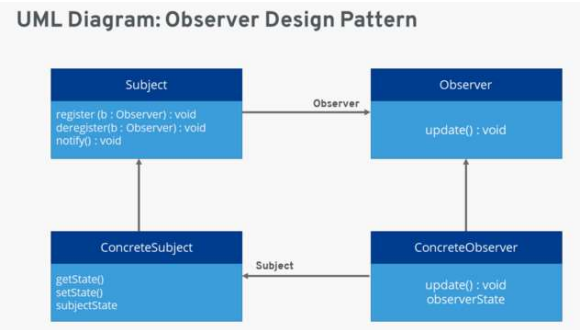
3. Design Pattern **Singleton** :

Il permet de s'assurer qu'une classe puisse posséder qu'une seule et unique instance sous peine de générer des erreurs. Le singleton pattern appartient à la catégorie des **patrons de création**. Son but est d'éviter qu'une classe ne crée plus d'un objet. Pour ce faire, on crée l'objet souhaité dans une classe propre, puis on le récupère sous forme d'instance statique.



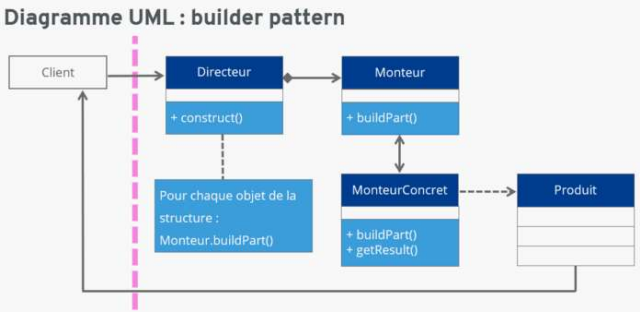
4. Design Pattern **Observer** :

On l'utilise pour limiter le couplage entre les modules aux seuls phénomènes à observer. Aussi, il permet une gestion simplifiée d'observateurs multiples sur un même observable. Ces derniers effectuent l'action adéquate en fonction des informations qui parviennent depuis des modules qu'ils observent.



5. Design Pattern **Builder** :

Ce DP fournit une interface pour créer des objets dans une superclasse, mais permet aux sous-classes de modifier le type d'objets qui seront créés. Il est principalement utilisé lorsque l'objet ne peut pas être créé en une seule étape comme dans la désérialisation d'un objet complexe raison pour laquelle nous le choisissons pour notre projet.



6. Design Pattern **Decorator** :

Un décorateur est le nom d'une des structures de patron de conception. Il permet d'attacher dynamiquement de nouvelles responsabilités à un objet. Les décorateurs offrent une alternative assez souple à l'héritage pour composer de nouvelles fonctionnalités.