REPUBLIQUE DEMOCRATIQUE DU CONGO
ENSEIGNEMENT SUPERIEUR ET UNIVERSITAIRE
UNIVERSITE LOYOLA DU CONGO
FACULTE D'INGENIERIE ULC-ICAM
DEPARTEMENT DE GENIE INFORMATIQUE

Troisième année de licence

DETAIL ET EXPLICATION DE TOUS LES DESIGN PATTERNS UTILISES DANS LE PROJET

GROUPE 2:

- BENJAMIN OLINABANJI
- SAMUEL BIRAHEKA
- GLORIA WASINGYA
- MICHEE KINSWEME

2024-2025

TABLE DES MATIERES

I. DEFINITION

Un **design pattern** (ou "patron de conception" en français) : est une solution générale réutilisable à un problème récurrent dans un contexte donné en développement logiciel. Il ne s'agit pas de code prêt à l'emploi, mais plutôt d'une description ou d'un modèle de solution qui peut être appliqué pour résoudre un problème particulier dans la conception de logiciels.

Les design patterns sont comme des "**recettes**" éprouvées qui aident les développeurs à structurer leur code de manière efficace, lisible et maintenable. Ils facilitent la communication entre développeurs en fournissant un vocabulaire commun et encouragent les bonnes pratiques de conception.

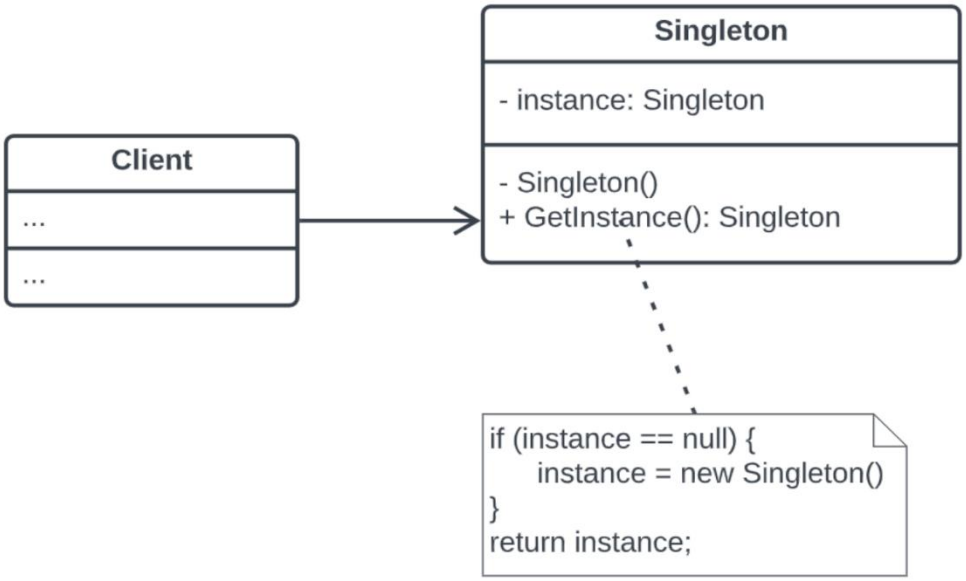
Caractéristiques clés d'un Design Pattern :

- Il peut être appliqué dans de nombreux projets différents.
- C'est une méthode ou une approche conceptuelle, plutôt qu'une implémentation directe.
- Chaque pattern est utile dans des situations ou des contextes spécifiques.

II. EXPLICATION DE TOUS LES DP

II.1. Le Singleton Pattern

Est un modèle de conception créatif qui vous permet de garantir qu'une classe n'a qu'une seule instance, tout en fournissant un point d'accès global à cette instance.



A. Concept du Singleton Pattern dans le modèle MVC

Dans une application utilisant l'architecture MVC (Model-View-Controller), le Singleton Pattern peut être appliqué pour des composants qui doivent être accessibles globalement et partagés à travers plusieurs contrôleurs et vues.

Voici comment le Singleton Pattern s'intègre dans l'architecture MVC :

- **Model**

Dans le contexte MVC, le Singleton peut être utilisé pour gérer des objets liés aux données et aux règles métier qui ne changent pas souvent, comme un cache en mémoire ou un gestionnaire de configuration qui contient les paramètres de l'application.

- **View**

Les vues ne nécessitent généralement pas l'utilisation du Singleton. Cependant, elles peuvent accéder à des services Singleton, comme un service de gestion de thèmes ou de langue, pour assurer un affichage cohérent.

- **Controller**

Les contrôleurs peuvent avoir besoin d'accéder aux services Singleton pour des opérations comme l'accès à la base de données, la gestion des sessions utilisateur ou l'enregistrement des logs.

B. Pourquoi utiliser le Singleton Pattern en MVC ?

1. Le Singleton est idéal pour gérer des ressources qui doivent être partagées, comme un gestionnaire de connexion à une base de données ou un service de cache.
2. Il est utile pour des composants qui contiennent des informations de configuration globales, comme des paramètres d'application ou des clés d'API.
3. En instanciant un objet une seule fois, le Singleton réduit la surcharge de création répétée de la même ressource, ce qui peut être crucial pour des composants gourmands en ressources.

C. Avantages

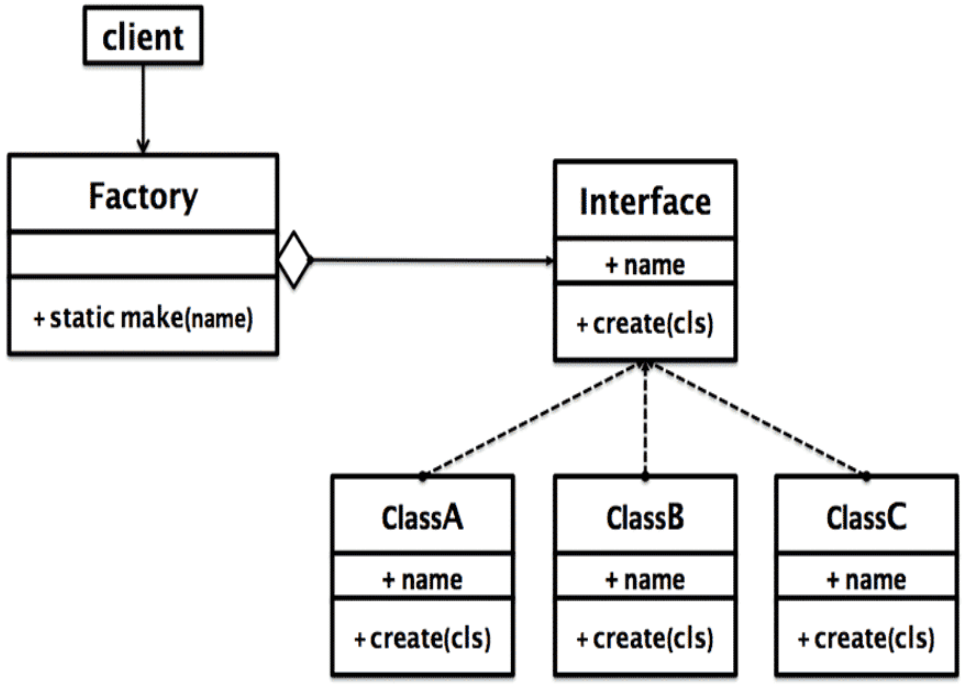
- Assure qu'une seule instance est créée, ce qui économise des ressources.
- Facile à accéder de n'importe où dans l'application.

D. Inconvénients

- Peut compliquer les tests unitaires, car il est difficile de remplacer ou de simuler un singleton.
- Le Singleton peut introduire des dépendances invisibles, ce qui rend le code plus complexe.

II.2. Le Factory Method

Est un modèle de conception créatif qui fournit une interface pour créer des objets dans une superclasse, mais permet aux sous-classes de modifier le type d'objets qui seront créés.



A. Concept du Factory Method Pattern dans le modèle MVC

Dans l'architecture **MVC (Model-View-Controller)**, le **Factory Method Pattern** est particulièrement utile pour créer des objets de manière dynamique et flexible. Cela peut être bénéfique pour des scénarios où vous devez instancier des objets de manière conditionnelle ou en fonction de paramètres.

Voici comment le Factory Method s'intègre dans une application MVC

- **Model**

Le Factory Method peut être utilisé pour créer des objets de modèle spécifiques selon des critères ou des paramètres, comme des objets qui interagissent avec une base de données.

- **Controller**

Les contrôleurs peuvent utiliser des (factories) pour obtenir des instances de services ou de composants métiers sans connaître les détails de leur création.

- **View**

Dans le contexte des vues, le Factory Method est rarement utilisé, mais il peut être appliqué pour générer des composants d'interface utilisateur spécifiques.

B. Pourquoi utiliser le Factory Method Pattern en MVC ?

1. Le Factory Method permet de centraliser la logique de création, ce qui simplifie le code du contrôleur.
2. Ajouter de nouveaux types d'objets ou de comptes devient facile, car il suffit de créer une nouvelle implémentation de la Factory.
3. Le contrôleur n'a pas besoin de connaître les détails des classes concrètes, ce qui rend le code plus flexible et plus maintenable.

C. **Avantages** :

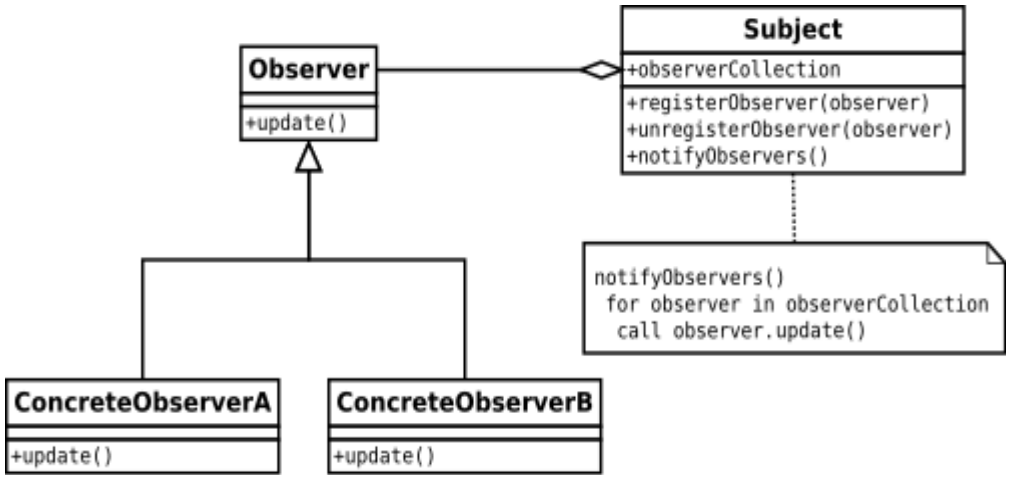
- Simplifie le code client en le dissociant de la logique de création.
- Facilite l'ajout de nouvelles classes sans modifier le code existant.

D. Inconvénients :

- Peut introduire plus de classes et de complexité.

II.3. L'Observer Pattern

Est un modèle de conception comportementale qui vous permet de définir un mécanisme d'abonnement pour notifier plusieurs objets de tous les événements qui se produisent sur l'objet qu'ils observent.



A. Concept de l'Observer Pattern dans le modèle MVC

Dans une application utilisant l'architecture **MVC (Model-View-Controller)**, l'Observer **Pattern** joue un rôle clé, notamment pour la gestion de la communication entre le **Model** et la **View**. Lorsque le modèle subit des changements, les vues qui dépendent de ce modèle doivent être mises à jour automatiquement, sans qu'il soit nécessaire de coupler étroitement ces composants.

- **Model**

Le modèle joue souvent le rôle de *sujet*. Il notifie les vues lorsque des changements surviennent dans les données.

- **View**

Les vues sont les *observateurs*. Elles se mettent à jour en réponse aux notifications provenant du modèle.

- **Controller**

Le contrôleur peut être impliqué pour gérer la logique entre les actions de l'utilisateur et les mises à jour des données du modèle, mais il n'interagit généralement pas directement avec l'Observer Pattern.

B. Pourquoi utiliser l'Observer Pattern en MVC ?

1. L'Observer Pattern permet de réduire le couplage entre les composants. Le modèle ne connaît pas les détails des observateurs, ce qui rend l'application plus flexible et maintenable.

2. Les vues ou autres composants qui dépendent des données sont automatiquement mis à jour lorsqu'un changement survient, simplifiant la gestion de l'état de l'application.
3. Il est facile d'ajouter ou de supprimer des observateurs sans modifier le modèle de base.

C. Avantages :

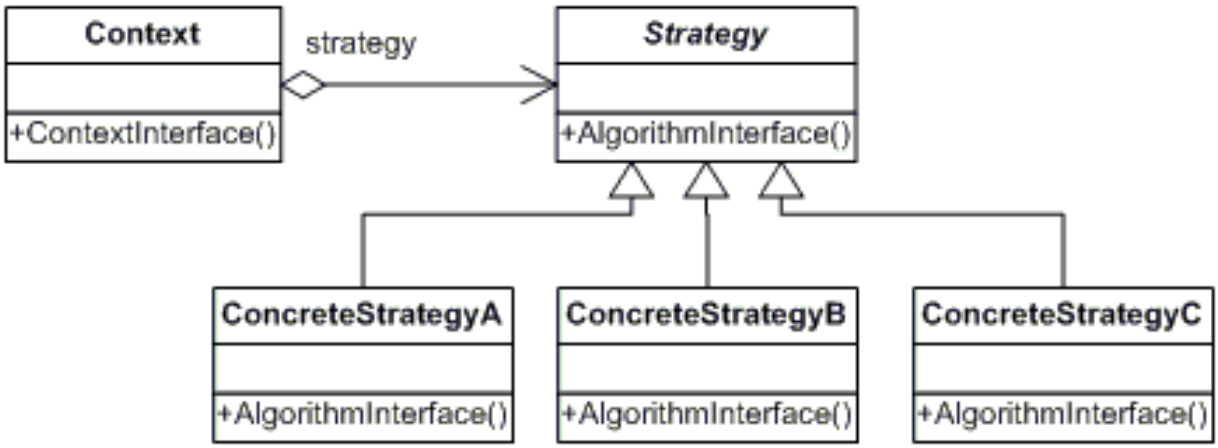
- Les observateurs et le sujet sont faiblement couplés, ce qui rend le code plus flexible.
- Les mises à jour automatiques simplifient la gestion de l'état.

D. Inconvénients :

- Peut devenir difficile à gérer lorsque le nombre d'observateurs augmente.
- Si trop d'observateurs doivent être notifiés fréquemment, la performance peut en souffrir.

II.4. Strategy Pattern

Est un modèle de conception comportementale qui vous permet de définir une famille d'algorithmes, de placer chacun d'eux dans une classe distincte et de rendre leurs objets interchangeables.



A. Concept du Strategy Pattern dans le modèle MVC

Dans une application utilisant l'architecture **MVC (Model-View-Controller)**, le **Strategy Pattern** peut être utilisé pour organiser et rendre flexible la logique métier qui doit être interchangeable ou configurable, comme des calculs de tarifs, des validations, ou des comportements d'affichage.

- **Model**

Le modèle peut contenir la logique qui utilise les différentes stratégies pour gérer les données.

- **Controller**

Le contrôleur joue souvent le rôle de coordinateur, choisissant la stratégie appropriée en fonction des entrées de l'utilisateur ou des paramètres de l'application.

- **View**

La vue n'implémente généralement pas le Strategy Pattern, mais elle peut dépendre indirectement de la stratégie sélectionnée pour ajuster l'affichage.

B. Pourquoi utiliser le Strategy Pattern en MVC ?

1. Le Strategy Pattern permet de modifier la logique métier sans toucher au code des classes qui utilisent cette logique, ce qui est particulièrement utile pour des algorithmes complexes ou pour des calculs qui peuvent varier.
2. Chaque algorithme ou logique de traitement est encapsulé dans sa propre classe, ce qui facilite la maintenance et le test unitaire.
3. Il est simple d'ajouter de nouvelles stratégies sans modifier le code existant. Vous pouvez ajouter de nouvelles classes qui implémentent l'interface de stratégie.

C. Avantages

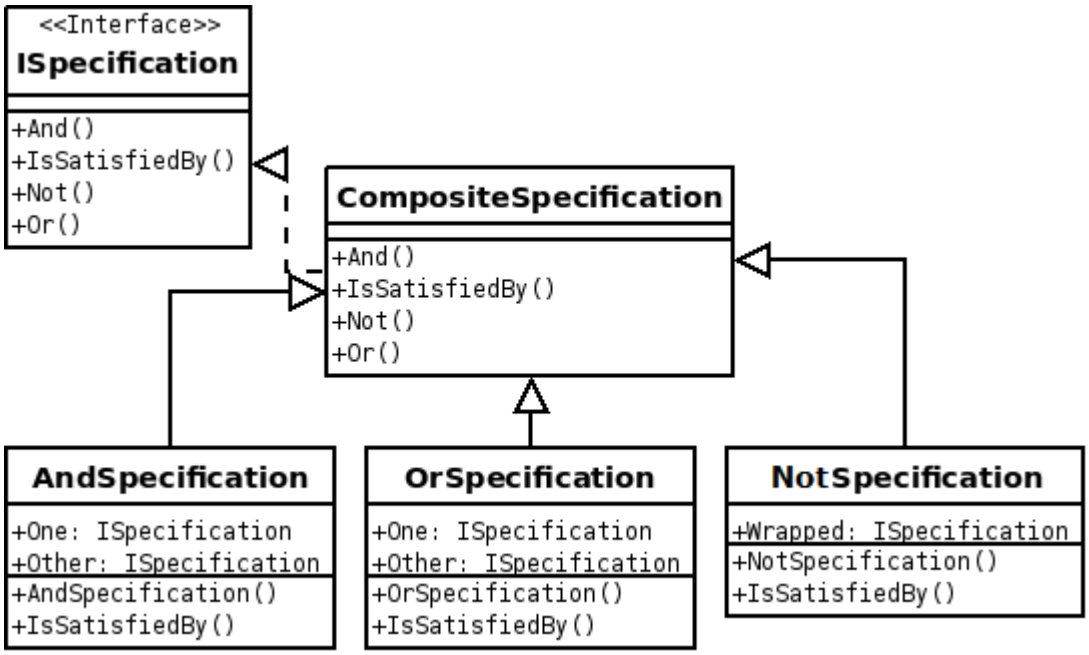
- Le pattern sépare la logique de traitement des algorithmes de la classe qui les utilise, améliorant ainsi la lisibilité et la modularité du code.
- Les stratégies peuvent être réutilisées dans différentes parties de l'application ou même dans d'autres projets.
- L'ajout de nouvelles stratégies est facile et ne nécessite pas de modifier le code du client, ce qui rend l'application évolutive.

D. Inconvénients

- L'utilisation de plusieurs classes de stratégie peut rendre le code plus complexe, surtout pour des projets de petite envergure.
- Si l'injection de dépendances n'est pas utilisée correctement, il peut être difficile de gérer la création et l'utilisation des stratégies.
- Si les stratégies sont nombreuses et complexes, le choix de la stratégie appropriée peut introduire un surcoût de performance.

II.5. Repository Pattern

C'est un design pattern utilisé pour centraliser la logique d'accès aux données. Il permet de créer une couche intermédiaire entre la logique métier et la couche de persistance, facilitant ainsi la gestion des opérations de base de données (CRUD) Tout en rendant l'application plus modulaire et testable.



A. Concept du Repository Pattern dans le modèle MVC

Dans une architecture **MVC (Model-View-Controller)**, le **Repository Pattern** est souvent utilisé pour gérer les interactions avec la base de données de manière organisée et découplée. Il offre un niveau d'abstraction qui permet de séparer la logique métier (dans le contrôleur) de la logique de persistance des données (qui réside dans le dépôt).

- **Model** : Représente les entités ou les objets métiers qui sont utilisés pour manipuler les données.
- **Repository** : Gère la logique d'accès aux données pour les entités. Cela inclut les méthodes pour récupérer, ajouter, mettre à jour, et supprimer des données.
- **Controller** : Le contrôleur utilise les dépôts pour accéder aux données et les fournir aux vues sans gérer directement les interactions avec la base de données.

B. Pourquoi utiliser le Repository Pattern dans MVC ?

1. Le Repository Pattern sépare la logique d'accès aux données de la logique métier du contrôleur, rendant le code plus propre.

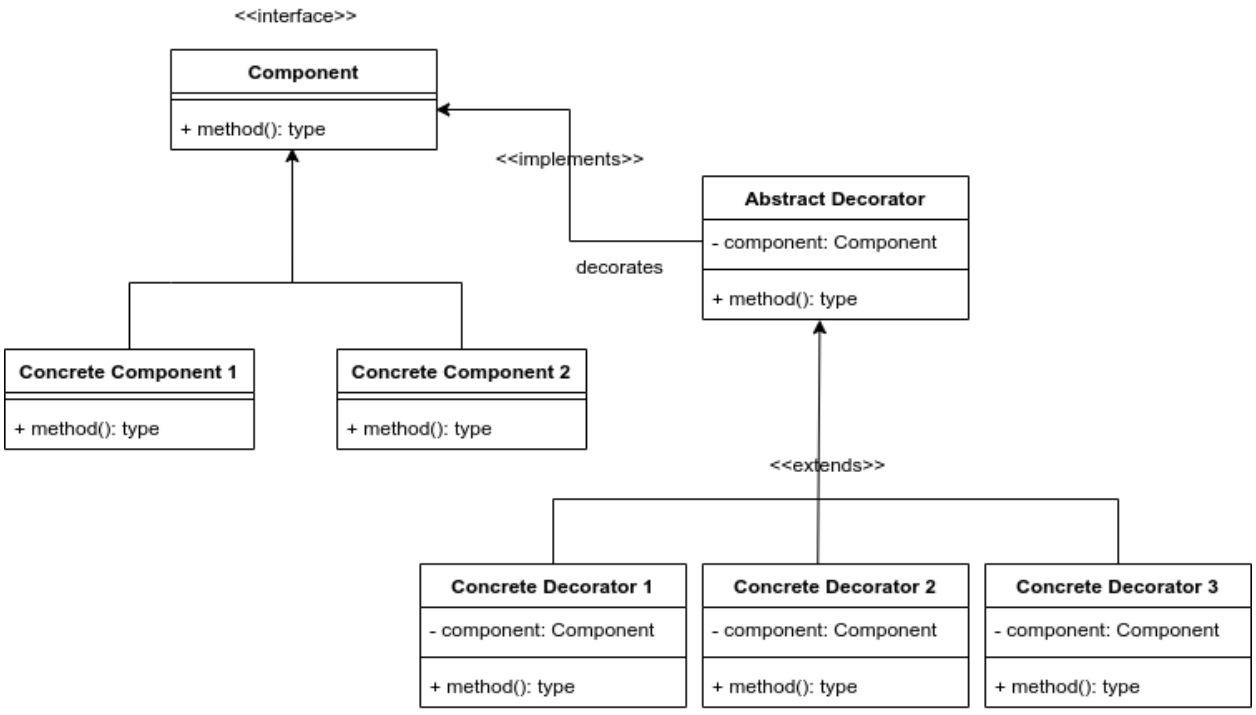
2. Si vous décidez de changer la technologie de persistance (par exemple, de Entity Framework à une API de base de données NoSQL), vous n'avez qu'à modifier le dépôt, sans toucher au reste de l'application.
3. Il devient plus facile de tester la logique des contrôleurs sans avoir à accéder à la base de données, en utilisant des dépôts simulés.

C. Avantages du Repository Pattern

1. Le contrôleur n'a pas besoin de connaître les détails de la persistance des données, ce qui rend le code plus propre et plus facile à maintenir.
2. Grâce à l'abstraction fournie par le Repository Pattern, il est plus facile de tester la logique métier en utilisant des dépôts simulés (mocks) ou des faux objets (fakes).
3. Le code d'accès aux données peut être réutilisé dans plusieurs endroits de l'application, réduisant ainsi les duplications.

II.6. Decorator Pattern

C'est un design pattern structurel qui permet d'ajouter dynamiquement des fonctionnalités ou des comportements à un objet, sans modifier sa structure d'origine. Cela se fait en enveloppant l'objet original avec des classes "décorateurs" qui implémentent des fonctionnalités supplémentaires.



A. Concept du Decorator Pattern

- **Objectif** : Le but principal du Decorator Pattern est d'ajouter des responsabilités à un objet de manière flexible et extensible. Ce pattern est souvent utilisé en remplacement de l'héritage lorsqu'il est préférable de composer des fonctionnalités dynamiquement plutôt que de les intégrer via une hiérarchie de classes.
- **Principe de fonctionnement** : Le Decorator Pattern utilise des classes "décorateurs" qui implémentent la même interface que l'objet qu'elles décorent. Les appels de méthodes sont généralement transmis à l'objet encapsulé, mais les décorateurs peuvent ajouter de nouvelles fonctionnalités avant ou après le passage de l'appel.

B. Comment le Decorator Pattern s'intègre dans le modèle MVC

Dans une architecture **MVC (Model-View-Controller)**, le Decorator Pattern est souvent utilisé pour ajouter des fonctionnalités aux modèles ou aux vues. Voici quelques exemples d'utilisation :

- **Dans le Model** : Le pattern peut être utilisé pour enrichir un objet modèle avec des fonctionnalités supplémentaires, comme des validations, des calculs ou des transformations de données.
- **Dans la View** : On peut utiliser le Decorator Pattern pour ajouter des styles ou des comportements aux éléments de l'interface utilisateur sans modifier la vue d'origine.
- **Dans le Controller** : Bien que ce soit plus rare, il est possible d'utiliser ce pattern pour ajouter des fonctionnalités de logging, d'authentification, ou de validation avant que le contrôleur ne traite les requêtes.

C. Cas d'utilisation courants en MVC

1. Ajouter des fonctionnalités comme la validation, la transformation de données, ou l'enrichissement des modèles.
2. Ajouter des styles ou des formats d'affichage à des objets de vue sans modifier le code des vues de base.
3. Ajouter des couches de sécurité, de journalisation, ou de validation des entrées avant de passer les requêtes au contrôleur.

D. Avantages

- Les fonctionnalités peuvent être ajoutées de manière dynamique, et plusieurs décorateurs peuvent être combinés pour créer de nouveaux comportements.
- Les décorateurs peuvent être réutilisés dans différentes parties de l'application, ce qui réduit la duplication de code.

- Le code est plus facile à maintenir et à comprendre, car chaque décorateur se concentre sur une fonctionnalité spécifique.

E. Inconvénients

- L'utilisation de plusieurs décorateurs peut rendre le code plus difficile à suivre, surtout si les décorateurs sont nombreux et imbriqués.
- L'enchaînement de plusieurs décorateurs peut entraîner une légère surcharge de performance, bien que cela ne soit généralement pas un problème majeur.