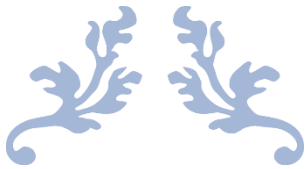

DESIGN PATTERN

Groupe 12

1. Factory (Fabrique)

Le **Factory Design Pattern** est un patron de création qui se concentre sur la création d'objets sans exposer leur logique d'instanciation au client. Au lieu de créer des objets directement à l'aide de leur constructeur, une classe Factory centralise cette responsabilité. Cela permet de :

- **Encapsuler** la logique de création d'objets.
- **Réduire** les dépendances entre le code client et les classes concrètes.
- **Faciliter** l'ajout de nouveaux types d'objets sans modifier le code client.

En résumé, le pattern Factory améliore la flexibilité et l'extensibilité en isolant la logique d'instanciation.

2. MVC (Model-View-Controller)

Le **MVC Design Pattern** est une architecture qui divise une application en trois composants principaux pour séparer les responsabilités :

- **Model** : Gère les données, la logique métier, et l'état de l'application. Il est indépendant de l'interface utilisateur.
- **View** : Représente la présentation visuelle des données (interface utilisateur). Elle interagit uniquement avec le modèle pour afficher les informations.
- **Controller** : Sert d'intermédiaire entre le modèle et la vue. Il gère les entrées de l'utilisateur, met à jour le modèle, et déclenche la mise à jour de la vue.

L'objectif principal du MVC est de **séparer les préoccupations** pour rendre l'application plus modulaire, testable, et maintenable.

3. Observer (Observateur)

Le **Observer Design Pattern** est un patron comportemental qui établit une relation entre un objet **Sujet** (ou observable) et plusieurs objets **Observateurs**.

- Lorsqu'un changement d'état survient dans le sujet, il **notifie automatiquement** ses observateurs.
- Les observateurs, à leur tour, réagissent en conséquence.

Ce pattern est particulièrement utile dans les systèmes réactifs ou dans des situations où une modification d'état doit être propagée à plusieurs parties du système, comme les interfaces utilisateur, les événements ou les flux de données.

4. Singleton

Le **Singleton Design Pattern** est un patron de création qui garantit qu'une classe ne peut avoir qu'une **seule instance** dans tout le programme. Cette instance est accessible globalement.

Principales caractéristiques :

- **Contrôle d'accès** : Centralise la gestion d'une ressource partagée (ex : une base de données, un logger).
- **Réduction des doublons** : Empêche la duplication de ressources coûteuses.
- **Point d'accès global** : Simplifie l'accès à l'instance unique, sans avoir à la passer explicitement.

Cependant, le Singleton peut entraîner des dépendances globales et compliquer les tests unitaires s'il n'est pas utilisé avec précaution

Différences clés entre ces patterns :

Pattern	Type	Objectif principal
Factory	Création	Encapsuler la logique de création pour permettre une extensibilité.
MVC	Architecture	Séparer les responsabilités pour une meilleure modularité.
Observer	Comportemental	Synchroniser automatiquement les objets en cas de changement d'état.
Singleton	Création	Garantir une instance unique pour gérer des ressources partagées.

Ces patterns combinés dans une application permettent de structurer le code de manière robuste, extensible et maintenable.