MAPC - CC 1

Tout ce qu'on a fait =>

CMs :

- MAPC 1
- MAPC 2
- MAPC 3
- MAPC 4
- MAPC 5

TDS :

- TD 1
- TD 2
- TD 3
- TD 4

Au controle =>

5 exo X 2 points/ex

1. **Décrire un anti-patron** (avec exemple)
2. **Décrire un patron** (avec exemple)
3. **Analyser un bout de code** pour identifier ce qui ne va pas + explications et corrections
4. **Transformer un texte en diagramme de classes** (UML + commentaires pour implémentation des patrons)
5. **Modifier un code pour appliquer un patron spécifié**

** On a le droit à une feuille A4 recto-verso avec nous, avec tout ce qu'on veut écrire pour nous aider **

———————————————————————————

Contenu des cours :

MAPC 1 : organisation et objectifs

MAPC 2 :

- **DRY (Don't Repeat Yourself)**,
- **KISS(Keep It Simple, Stupid!)**,
- **SOLID**(Single responsibility principle (SCP); Open/closed principle (OCP); Liskov substitution principle (LSP); Interface segregation principle (ISP); Dependency inversion principle (DIP))

- **Modélisation UML**

MAPC 3 :

- **Encapsulation**
- **Énumération**
- **Immutabilité**
- **Contrats**
- **ANTI PATRON :**
  - **InappropriateIntimacy**† : (éviter les getters / setters automatiques, conception de mauvaise qualité, abstraction non effective)
  - **PrimitiveObsession**† : (pas de correspondance au métier, couplage fort abstraction / implémentation, modification complexe facilité de corruption des données)

MAPC 4 :

- **Interface**
- **Factory methods**
- **PATRON** :
  - **ITERATOR** : exposer des collections en respectant l'encapsulation
  - **TEMPLATE METHOD** : algorithme avec étapes particulières raffinées, choix statique
  - **STRATEGY** : dépendance à une famille d'algorithmes, choix dynamique
  - **FACTORY METHOD** : interface de création d'objets
- **ANTI PATRON** :
  - **Switch Statement**† **:** Usage problématique de multiples structures conditionnelles.

MAPC 5 :

- **État :** état exprimable vs atteignable vs souhaitable état concret vs abstrait
- **Diagrammes d'états d'UML**
- **Absence de valeur**
- **ANTI PATRON** :
  - **SPECULATIVE GENERALITY**† **:** Ajouter du code pour des besoins futurs incertains.
  - **TEMPORARY FIELD**†**s :** Champs présents dans une classe mais

- utilisés seulement dans des cas particuliers
- **LONG METHOD† :** Refactoring pour réduire la longueur et complexité d'une méthode.
- **STUPID**

- **PATRON :**
  - **NULL object :** Remplacer null par un objet avec comportement par défaut.
  - **SINGLETON :** une seule valeur pour une classe
  - **FLYWEIGHT :** n valeurs uniques pour une classe
  - **STATE :** respect machine à états pour gérer les comportements dynamiques d'un objet.

Couples possibles entre un patron et un anti-patron :

1. Primitive Obsession - Factory Method

- **Problème** : Utilisation excessive de types primitifs ou génériques au lieu de classes spécifiques au métier.
- **Solution avec Factory Method** : Encapsuler la création d'objets spécifiques au métier dans une méthode ou une classe dédiée pour standardiser la construction et éviter l'utilisation de types primitifs.
  Exemple : Créer une AdresseFactory qui produit des objets Adresse valides.

2. Inappropriate Intimacy - Encapsulation/Immutabilité

- **Problème** : Les classes se connaissent trop intimement (ex. : exposition via des getters/setters non nécessaires).
- **Solution avec Encapsulation** : Restreindre l'accès aux données internes en cachant les détails d'implémentation.
- **Solution avec Immutabilité** : Rendre les objets immuables pour éviter des modifications inattendues.

3. Switch Statement - Strategy

- **Problème** : Une logique complexe est gérée avec des blocs if/else ou switch, ce qui enfreint le principe OCP (Open/Closed Principle).
- **Solution avec Strategy** : Remplacer les branches conditionnelles par des classes qui implémentent une interface commune pour gérer dynamiquement différentes logiques.

4. Speculative Generality - KISS (Keep It Simple, Stupid!)

- **Problème** : Ajout de fonctionnalités inutiles ou non demandées qui rendent le code difficile à maintenir.

- **Solution avec KISS** : Concevoir uniquement ce qui est nécessaire, éviter la sur-ingénierie.

5. Temporary Field - State

- **Problème** : Des champs ne sont utilisés que dans certains états d'un objet, rendant le code difficile à comprendre et à maintenir.
- **Solution avec State** : Décomposer les états de l'objet en classes distinctes et gérer dynamiquement les champs selon l'état actuel.

6. Long Method - Template Method

- **Problème** : Une méthode est trop longue et contient plusieurs étapes imbriquées, rendant le code difficile à lire et à maintenir.
- **Solution avec Template Method** : Décomposer la méthode en plusieurs étapes et les encapsuler dans des méthodes distinctes, certaines pouvant être redéfinies par les sous-classes.

7. Absence de valeur (null) - Null Object

- **Problème** : L'utilisation fréquente de null pour signaler l'absence de valeur conduit à des exceptions si le traitement de null est oublié.
- **Solution avec Null Object** : Remplacer null par un objet qui encapsule un comportement par défaut.