Résumé des patrons de conception

Composite

- *Quand?* Il permet de modéliser une structure arborescente
- *Pourquoi?* C'est une structure agile qui permet d'ajouter de nouveaux types de noeuds sans difficulté
- *Comment?* On utilise une interface commune à tous les noeuds. Cela permet à chaque noeud d'avoir juste une liste de noeud comme fils.
- *Exemples d'utilisation :* Formules, Langages à balises, Formules logiques, Systèmes de fichiers, JavaFX-FXML...

Template Method

- *Quand?* Plusieurs classes similaires partagent du code
- *Pourquoi?* Pour éviter de la redondance de code. Cela permet d'ajouter des classes similaires à moindre coût.
- *Comment?* On utilise une classe abstraite qui va contenir le code commun et des méthodes abstraites vont contenir le code spécifique.
- *Exemples d'utilisation* : VariadicFormula

Strategy

- *Quand?* Plusieurs classes similaires partagent du code
- *Pourquoi?* Pour éviter la redondance de code. Pour séparer les responsabilités dans deux classes différentes.
- *Comment?* On délègue à un attribut de la classe une part des méthodes de cette classe, ces méthodes forme une interface qu'implémentera l'attribut.
- *Exemples d'utilisation* : VariadicFormula-> Operator, Treasure-> TreasureGenerator

Visitor

- *Quand?* Vous voulez ajouter un service à un ensemble de classes qui partagent une interface.
- *Pourquoi?* Pour ne pas modifer (trop) ces classes et cette interface.
- *Comment?* On rend ces classes visitable (accept qui apparait aussi dans l'interface) et on implémente le nouveau service dans une classe visitor qui a une méthode visit pour chaque classe de l'interface
- *Exemples* d'utilisation : Calcul d'un salaire pour des personnes occupant différentes fonction, ajout d'un service de serialisation, ajout d'un service de visualisation.

State

- *Quand?* Vous avez plusieurs états dans votre systême en fonction des interactions précédente. Votre systême est dynamique.
- *Pourquoi?* Cela vous permet de mieux partitionner les comportements en fonction de l'état et de rajouter de nouveaux états facilement.
- *Comment?* En déléguant à un attribut (state) l'execution du code de votre systême. Cet état implémente un interface et pourra être remplacé par un autre état au cours de l'éxécution.
- *Exemples d'utilisation* : Gestion d'un systême avec interface utilisateur, algorithme nécessitant plusieurs états, jeu de la vie.

Factory

- *Quand?* Vous remplacerez potentiellement une classe par une autre qui rend le même service.
- *Pourquoi?* Pour éviter de devoir modifier tous les appels au contructeurs lorsque vous ferez ce changement.
- *Comment?* En créant une méthode create pour chacun des objets que vous voulez créer dans une classe factory et en remplaçant tous les appels à un contructeur par un appel à la méthode create correspondante.
- *Exemples d'utilisation* : FormulaFactory, ButtonFactory

Abstract Factory

- *Quand?* Vous voulez pouvoir passer une Factory à une autre facilement
- *Pourquoi?* Cela vous évite de devoir modifier des factory déjà faîtes.
- *Comment?* En créant une interface AbstractFactory qui contient tous les create necessaire et chaque Factory implémentera cette interface
- *Exemples d'utilisation* : FormulaFactory, ButtonFactory

Builder

- *Quand?* Vous créez un objet complexe avec beaucoup d'attributs.
- *Pourquoi?* Pour décharger la classe de la responsabilité de création de cet objet qui n'est pas en tant que tel un service de cette classe.
- *Comment?* En créant un classe qui initialise les champs de l'objet (set) et qui une fois que les champs sont remplis crée l'objet (build).
- *Exemples d'utilisation* : Personnes, Notifications Android

Apdapter

- *Quand?* Vous avez une classe qui fournit sensiblement les services demandés mais qui n'implémente pas l'interface que vous voulez.
- *Pourquoi?* Pour éviter de modifier la classe existante;
- *Comment?* En créant une nouvelle classe qui a en attribut la classe existante et qui implémente l'interface voulue et adapte ses méthodes en appelant les méthodes de la classe en attribut
- *Exemples d'utilisation* : EnglishPainter

Decorator

- *Quand?* Vous voulez customiser les services d'un ensemble de classes qui partagent une interface.
- *Pourquoi?* De manière à avoir des variations très modulaires de classes
- *Comment?* En utilisant une classe abstraite qui implémente cette interface et qui a en attribut un élement de cette classe.
- *Exemples d'utilisation* : Verbose, debug

Observer

- *Quand?* L'appel de méthodes sur un objet va générer des modifications dans un certains nombre d'autres classes.
- *Pourquoi?* On veut déléguer à ces autres classes la façon de gérer ces modifications et garder une liberté sur le sobjets affectés.
- *Comment?* On utilise une classe abstraite qui va gérer une liste d'observer et leur transmettre les modifications de l'objet.
- *Exemples d'utilisation* : Interface graphique-> repaint,

ChainOfResponsability

- *Quand?* Vous voulez faire un switch sur des valeurs ou bien que vous voulez faire des if then else en série
- *Pourquoi?* Pour permettre une gestion modulaire de ce switch.
- *Comment?* En utilisant une classe abstraite qui gère le passage d'un handler à un autre.
- *Exemples d'utilisation* : Message d'erreurs, gestion de cas compliqués.

Proxy

- *Quand?* Quand vous voulez limiter les méthodes utilisables pour une classe.
- *Pourquoi?* Vous ne voulez pas modifier la classe existante et utiliser le code existant autant que possible.
- *Comment?* Vous ajoutez une classe qui a pour attribut un objet de la classe initiale et chaque méthode de cette classe renvoie sous le résultat de la méthode sur la classe initiale, soit une exception si vous ne voulez pas que la méthode soit utilisable.
- *Exemples d'utilisation* : Unmodifiable stack

Command

- *Quand?* Vous souhaitez pouvoir revenir en arrière sur des opérations.
- *Pourquoi?* Pour ne pas modifier le code des opérations.
- *Comment?* En ajoutant une classe qui va retenir l'ensemble des opérations faîtes dans une liste et qui va permettre de défaire ces opérations.
- *Exemples d'utilisation* : Calculatrice.

Classement

Organisation

- Template Method
- Strategy
- Composite
- Visitor
- State

Recyclage

- Adapter
- Proxy
- Command

Mechanisme

- Observer
- Decorator
- Chain of Responsability

Construction

- Factory
- Abstract Factory
- Builder
- Singleton