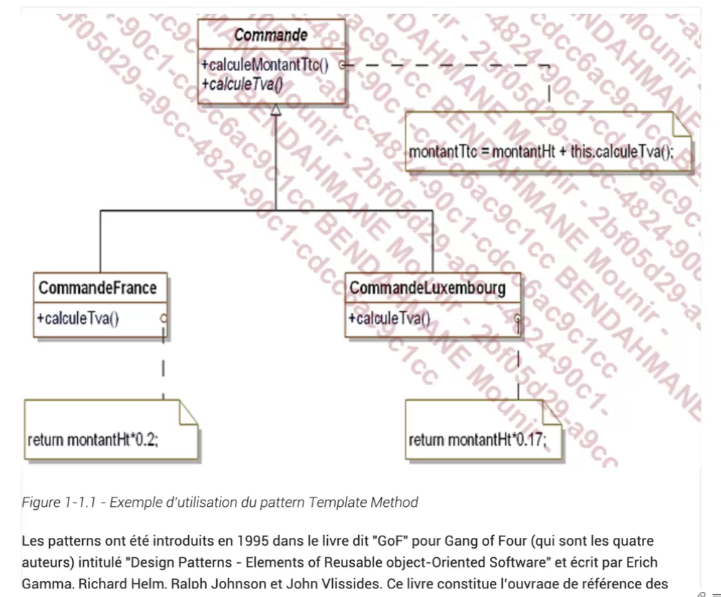
Cours

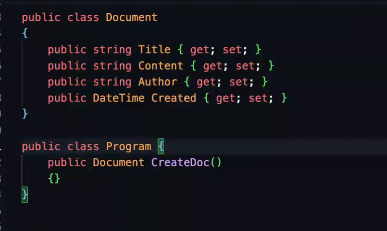
lundi 30 décembre 2024

09:15

* Patterns : schéma d'objets qui forme une solution a un problème connu et fréquent
* Décrit par des classes et de relation
* Dans le cadre de la POO
* Bases sur les bonnes pratiques

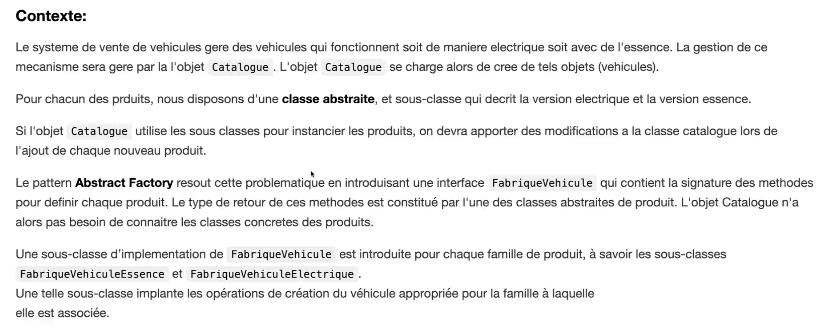
C'est un schéma UML (demander à l'exam)

* 

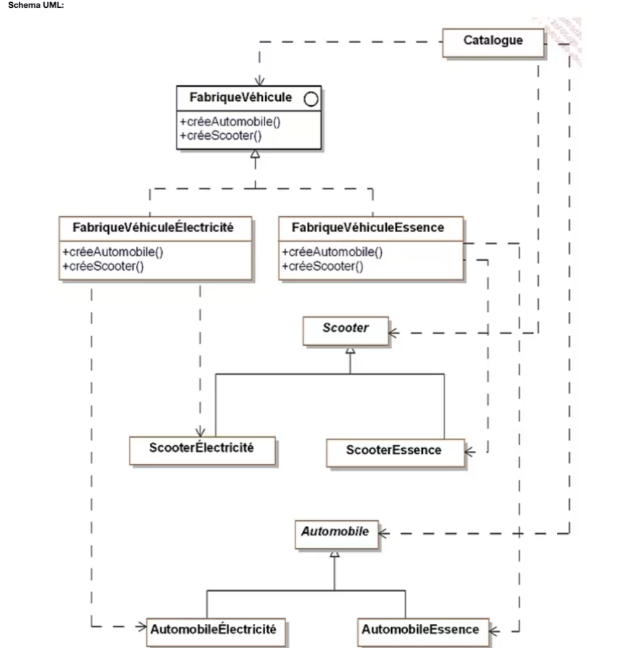
* La description des patterns : le langage UML et le C# (java …..)
* Pour chaque pattern :
  + Nom
  + Description
  + Exemple sous forme UML
  + Le cas d'utilisation
  + Un exemple de code en C#
* Une étude de cas pour la vente des véhicules en ligne :
  + Cahier de charges :
    - Véhicules destinée à la vente
    - Catalogue
    - Option
    - Panier/ gestion de commandes
* Patterns de construction : le mécanisme de création d'objets.
* Le but de pattern est la création d'objets regroupes en familles sans devoir connaitre les classes concrètes destinées à la création de ces objets
* Construction leur but d'abstraire la mécanisme de création d'objets. donc un système qui utilise un pattern de construction. il est indépendant de la façon dont les objets sont créés. Il favorise l'utilisation d'interface pour décrire les relations entre les objets. une interface permet d'établir des relations entre des objets
* En C# une instruction de création de classe :
  + Var objet = new MyClass(); // création d'une instance de la classe Myclass
* Dans certains cas il est nécessaire de paramétrer la construction d'objet
  + 

Ce code n'est pas bon pour le long terme

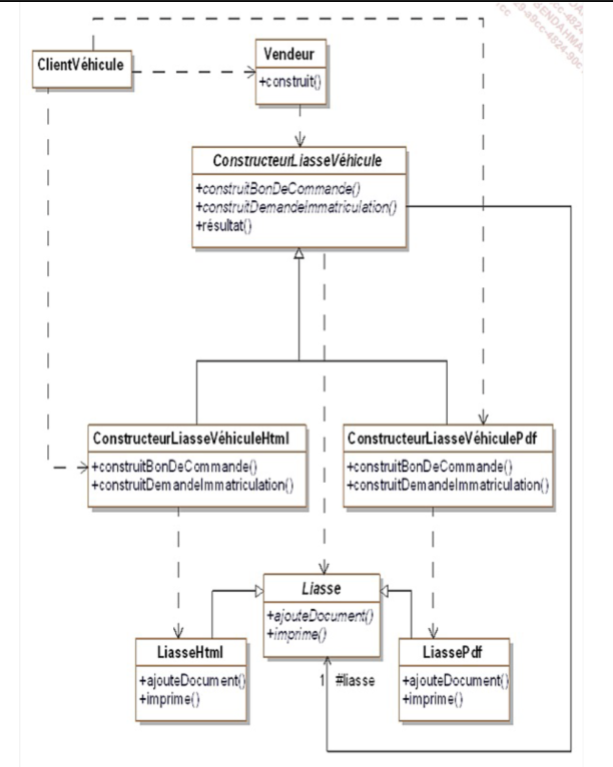
* 

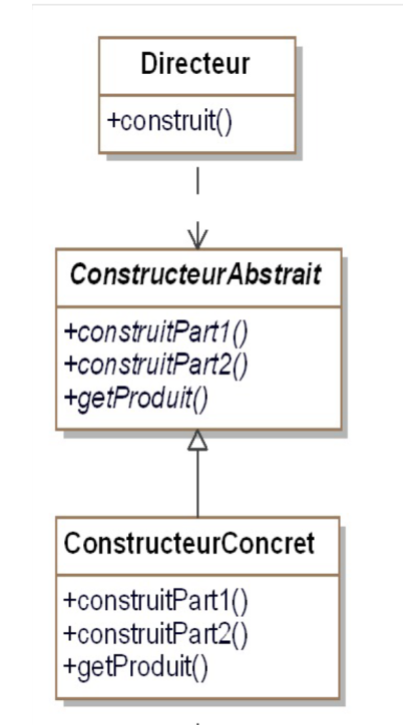
* 

* 

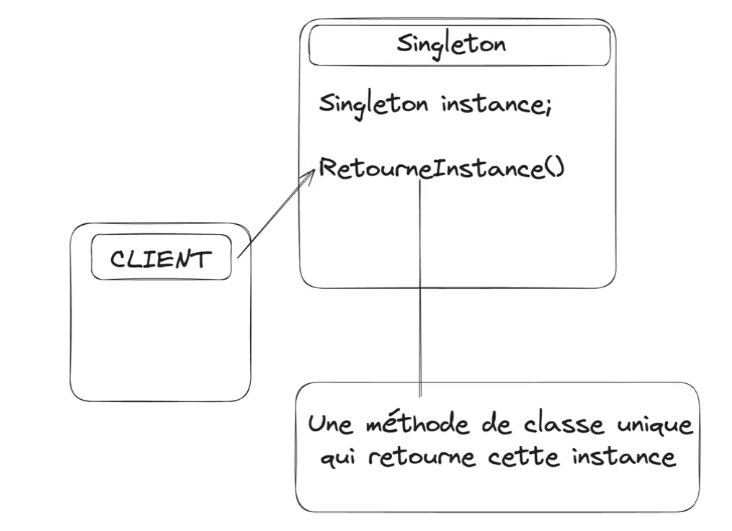
* 
* Fabrique véhicules c'est l'interface

* Commande :
  + Dotnet build
* **Pattern builder :**

* 

* 

* Pour le **pattern singleton :** permet d'assurer qu'une classe ne possède qu'une seule instance au cours de l'exécution du programme
* Une méthode de classe unique qui retourne cette instance

* 

* **Prototype :** création de nouveaux objets par duplication d'objet existants, c'est de prototype
* Capacité de clonage

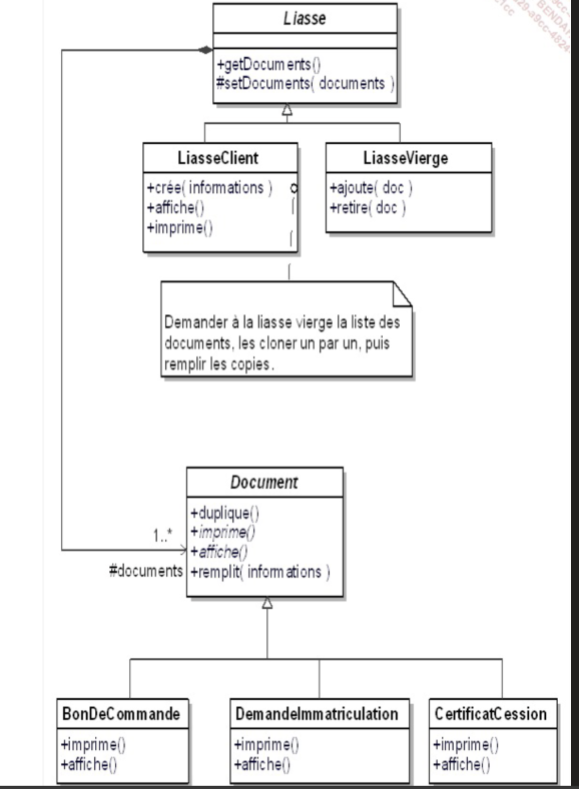
* Schéma pour le prototype :le design patterns prototype permet de créer des copies d'objet existants sans rendre le code dépendant de leurs classes concrètes , au lieu de créer un objet à partir de zéro, on clone un objet existant ( le prototype) et on modifie si besoin

c'est comme faire une photocopie d'un document : on copie l'original et on modifie la copie

l'avantage principale est la création d'objets complexe simplifiée

Cas d'usage:

* Objets avec de nombreuses configuration (éditeur graphique : formes géométriques)
* En jeux vidéo , création d'ennemis et leur variation en grand qualité
* Création d'une configuration par défaut dans un jeu/logiciel (permet de restaurer le défaut rapidement et de configurer a nouveau)
* Pour éviter la création d'objet système couteux : connexion réseaux , ressources partagés

* 

* **Patterns de structuration :** ça facilite l'Independence de l'interface d'un objet vis-à-vis de son implémentation. Dans le cas d'un ensemble d'objet, il s'agit de rendre cette interface indépendant de la hiérarchie des classes et de la composition des objets.

Encapsulation de la composition des objets

Les patterns de structuration encapsulent la composition des objets (augmentation de niveau d'abstraction)

Précédemment les patterns de création encapsulaient la création des objets

* **Pattern adapter:** convertir l'interface d'une classe existante en une interface attendue par ses clients (afin qu'ils puissent travailler ensemble )

Autrement dit c'est de fournir une nouvelle interface pour répondre aux besoins de clients

* **Pattern bridge :** séparer l'aspect d'implémentation d'un objet de son aspect représentation et l'interface

* **Pattern decorator :** le but du pattern est d'ajouter dynamiquement des fonctionnalités supplémentaires à un objet

Aucune modification de l'interface de l'objet

Transparent vis-à-vis des clients

Une alternative à la création d'une sous classe pour enrichir un objet

* **Pattern :** objectif c'est de fournir des solution pour distribuer les traitements et les algorithmes entre les objets

* **Pattern chain of responsibality :** construit une chaine d'objet telle que si un objet de la chaine ne peut pas répondre à la requête,   
  il puisse la transmettre à son successeur jusqu’à ce que l'un des objets répondent

Une chaine d'objet doit gérer une requête selon un ordre défini dynamiquement

La façon dont une chaine d'objets gère une requête ne doit pas être connue par ses clients

* **Pattern command :** permet de transformer une requête en un objet,

ce qui facilite les opérations comme les annulation ou les "retour en arrière "

ou encore les mises en file d'attente des demandes et leur suivi

diagramme UML illustre un **design pattern Abstract Factory** appliqué au contexte de fabrication de véhicules (voitures et scooters) avec deux variantes principales : **électrique** et **essence**.

**1. Abstract Factory (FabriqueVéhicule)**

* Classe de base abstraite.
* Déclare deux méthodes principales :
  + **créeAutomobile()** : crée un objet de type automobile.
  + **créeScooter()** : crée un objet de type scooter.

Elle définit une interface commune pour toutes les fabriques concrètes.

**2. Fabriques concrètes**

* **FabriqueVéhiculeElectricité** :
  + Implémente les méthodes de **FabriqueVéhicule** pour produire des **véhicules électriques**.
  + Retourne des instances de **AutomobileElectricité** et **ScooterElectricité**.
* **FabriqueVéhiculeEssence** :
  + Implémente les méthodes de **FabriqueVéhicule** pour produire des **véhicules à essence**.
  + Retourne des instances de **AutomobileEssence** et **ScooterEssence**.

**3. Produits (Products)**

Ces produits représentent les objets concrets que la fabrique produit :

* **Scooter** (classe abstraite) : sert de base pour les types de scooters :
  + **ScooterElectricité** : Scooter électrique.
  + **ScooterEssence** : Scooter à essence.
* **Automobile** (classe abstraite) : sert de base pour les types d’automobiles :
  + **AutomobileElectricité** : Automobile électrique.
  + **AutomobileEssence** : Automobile à essence.

**4. Catalogue (client)**

* Utilise une fabrique concrète (**FabriqueVéhiculeElectricité** ou **FabriqueVéhiculeEssence**) pour fabriquer les produits sans avoir besoin de connaître leur implémentation exacte. Cela garantit une indépendance entre le **client** (le code du catalogue) et les classes concrètes.

**5. Avantages (à mentionner dans le cours)**

* **Découplage** : Le client (Catalogue) est découplé des classes concrètes.
* **Cohérence** : Une fabrique garantit que tous les objets créés appartiennent à la même famille (ex. : véhicules électriques).
* **Extension facilitée** : Si une nouvelle famille de véhicules est ajoutée (par ex., hydrogène), il suffit d’ajouter une nouvelle fabrique concrète et ses produits.