GÉNIE LOGICIEL ORIENTÉ OBJET (GLO-2004)
ANALYSE ET CONCEPTION DES SYSTÈMES ORIENTÉS OBJETS (IFT-2007)

Automne 2016

Module 16 - Patrons de conception (partie 2)

Martin.Savoie@ift.ulaval.ca

Bachelier Génie logiciel, Chargé de cours,
département d'informatique et de génie logiciel

UNIVERSITÉ LAVAL

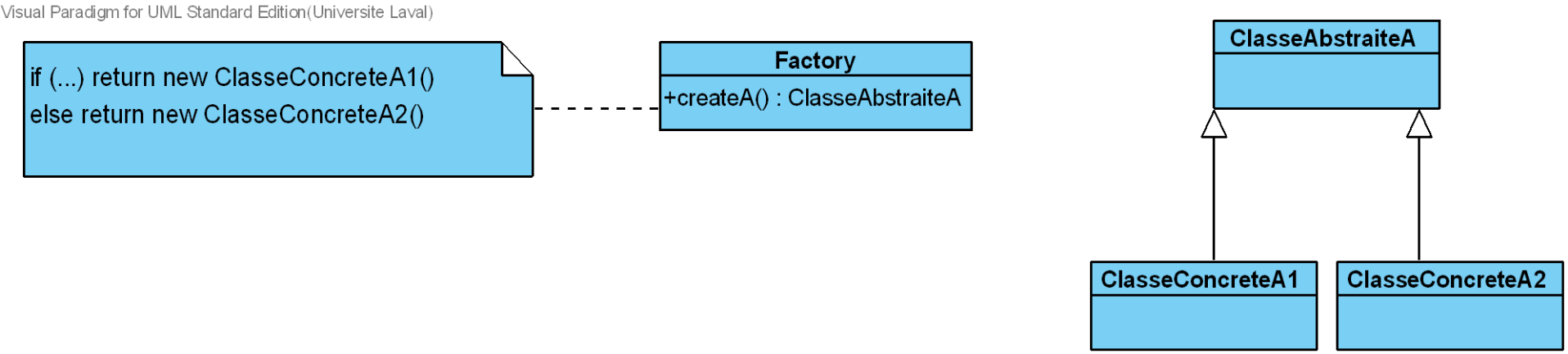
Factory
(Version simplifiée de AbstractFactory (Gof))

- « Factory » veut dire « Usine »
- **Idée générale**: quand j'ai besoin d'un objet, plutôt que de le créer moi-même en faisant un « objet = new ClasseQueJeVeux() » on délègue la responsabilité à une usine à objets:

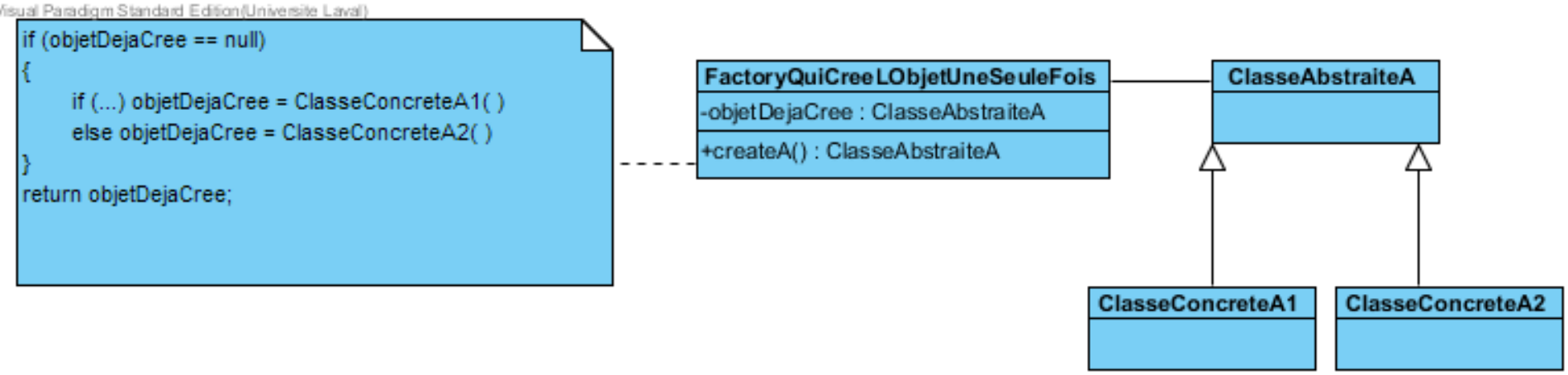
  objet = usine.nouvelObjetQuiRepondraAMonBesoinSVP(…);

- Autrement dit, ce patron offre une interface permettant de créer des objets sans avoir à connaître/spécifier leur classe concrète.

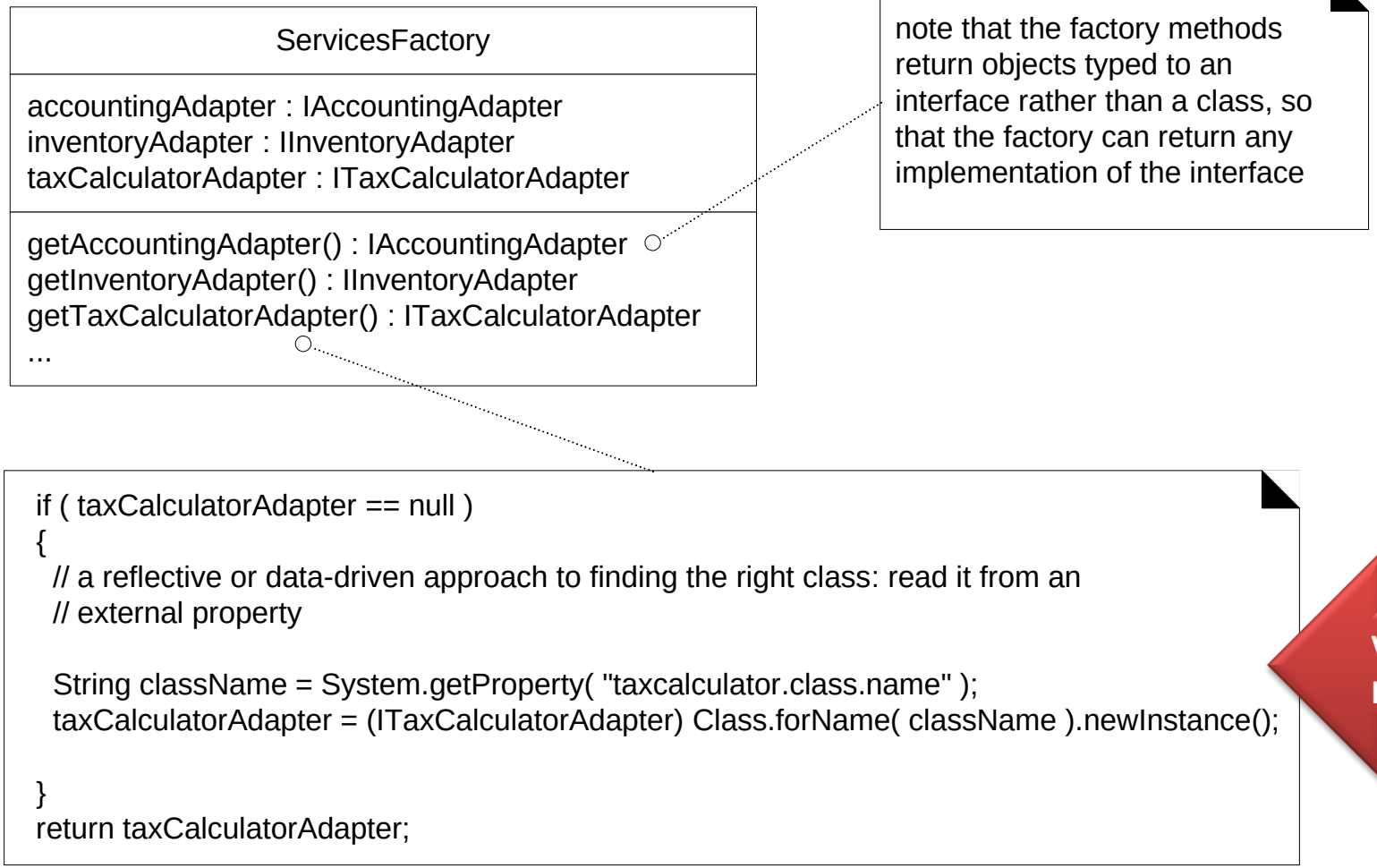
Factory de base



Factory qui crée l'objet une seule fois



NexGen POS

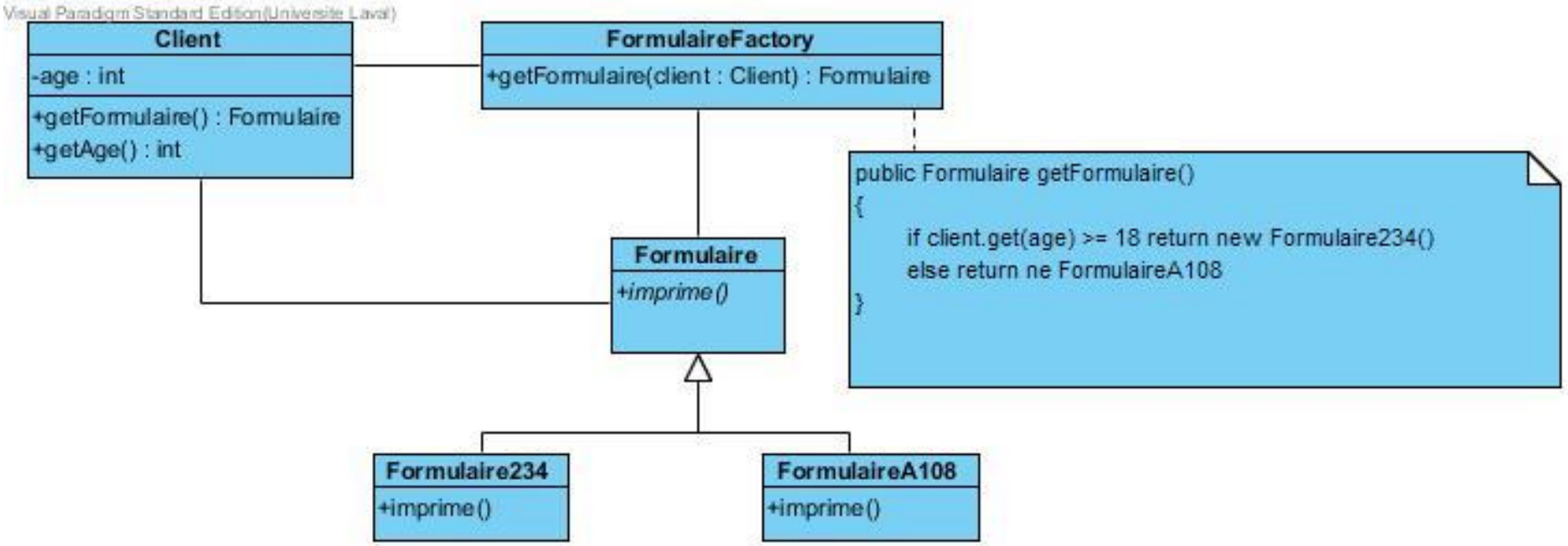


Exercice - Factory

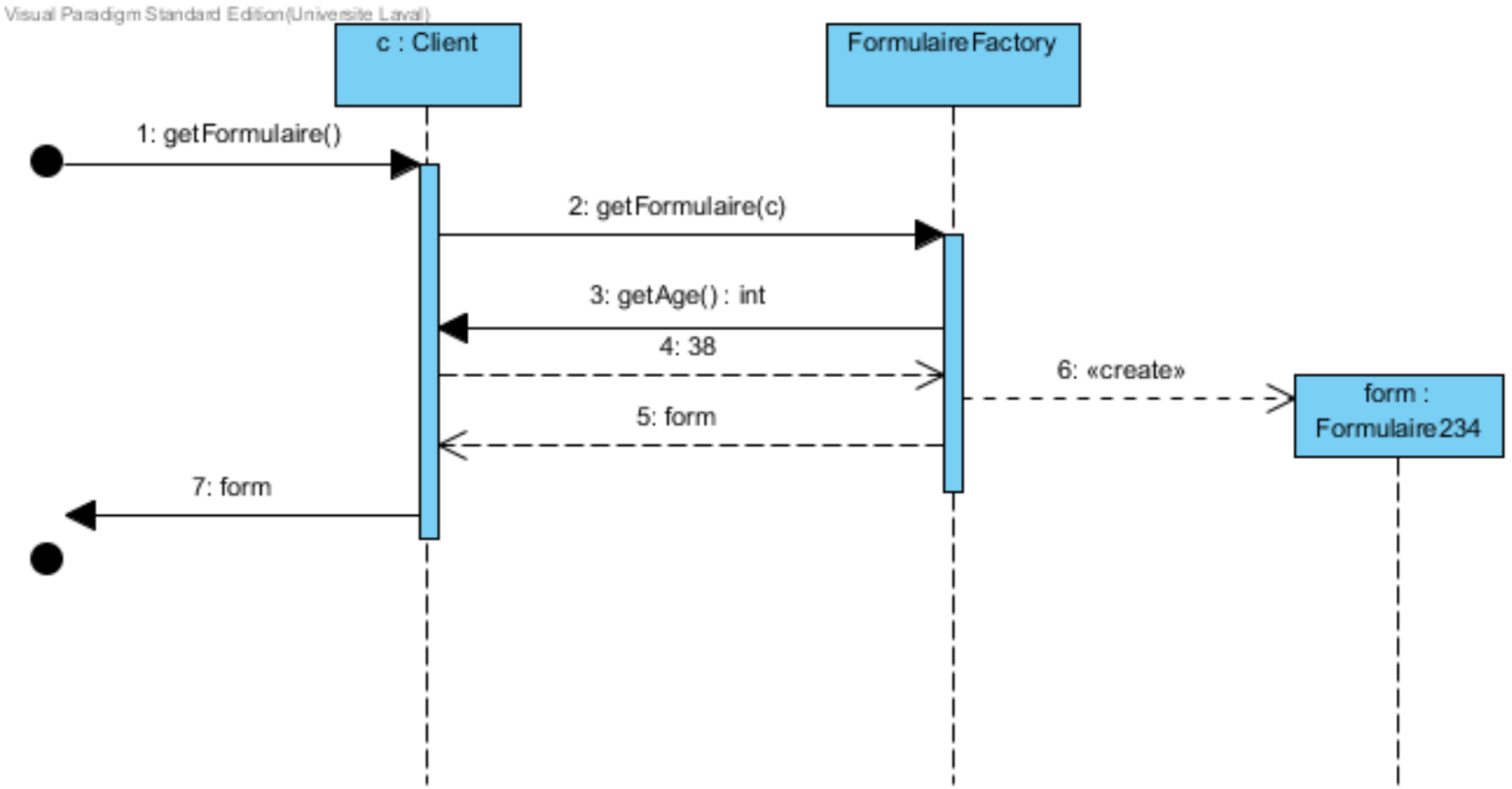
- Une classe appelée **Client** dispose d'un attribut « age ». On a besoin d'un objet **Formulaire** pour ce client.
- Dépendamment de l'âge du **Client** ce peut être un formulaire différent qui est retourné par : **Formulaire234** pour les personnes majeures ou le **FormulaireA108** pour les autres (deux classes descendantes de Formulaire).
- Comme il se peut qu'il y ait un jour plusieurs autres types de formulaires, on voudrait créer une classe **FormulaireFactory** qui crée le bon formulaire en fonction du Client.
- Tracez le diagramme de classes de conception
- Tracez un diagramme de séquence qui montre l'usage qui serait fait de la factory par la classe **Client**

Exercice - Factory

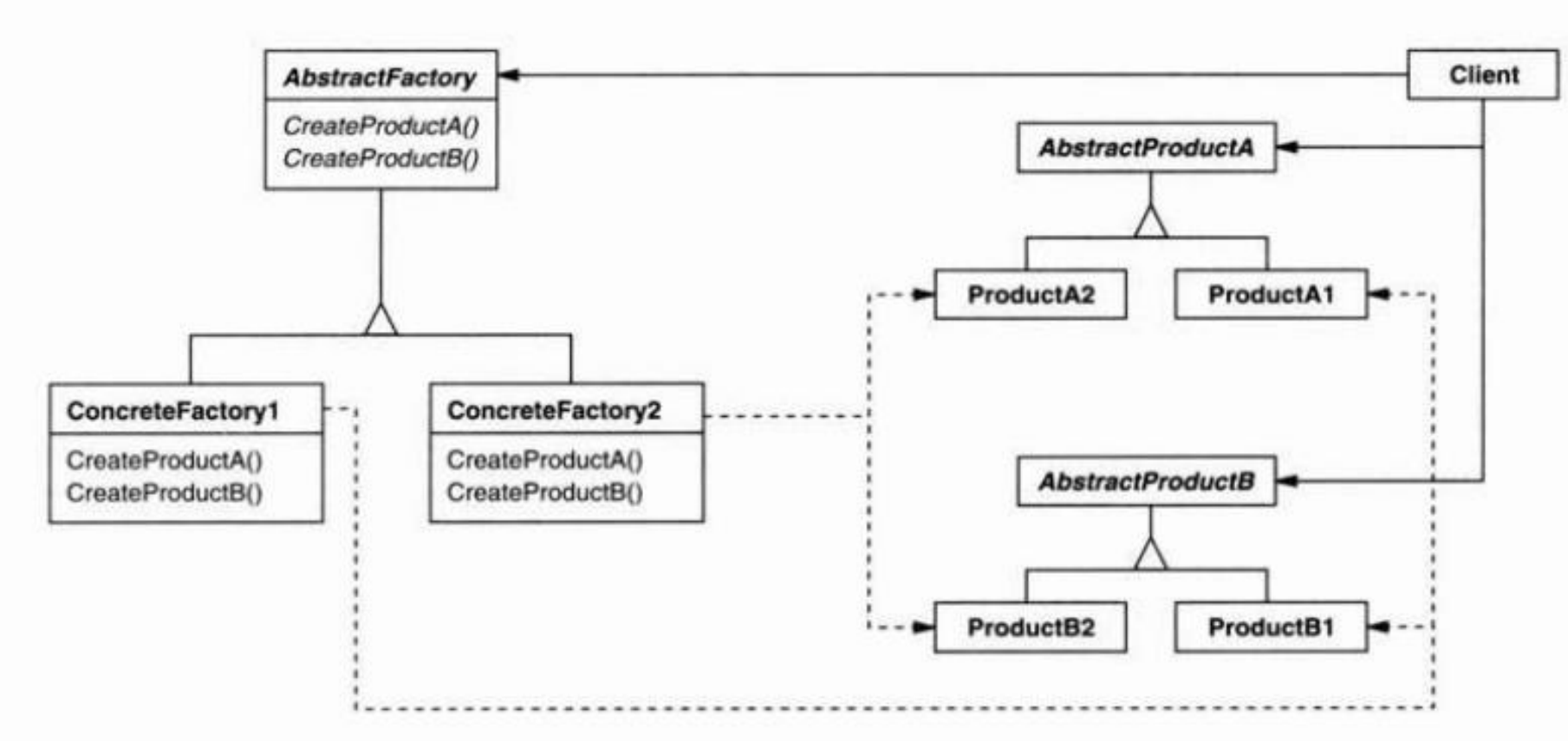
Solution - Factory



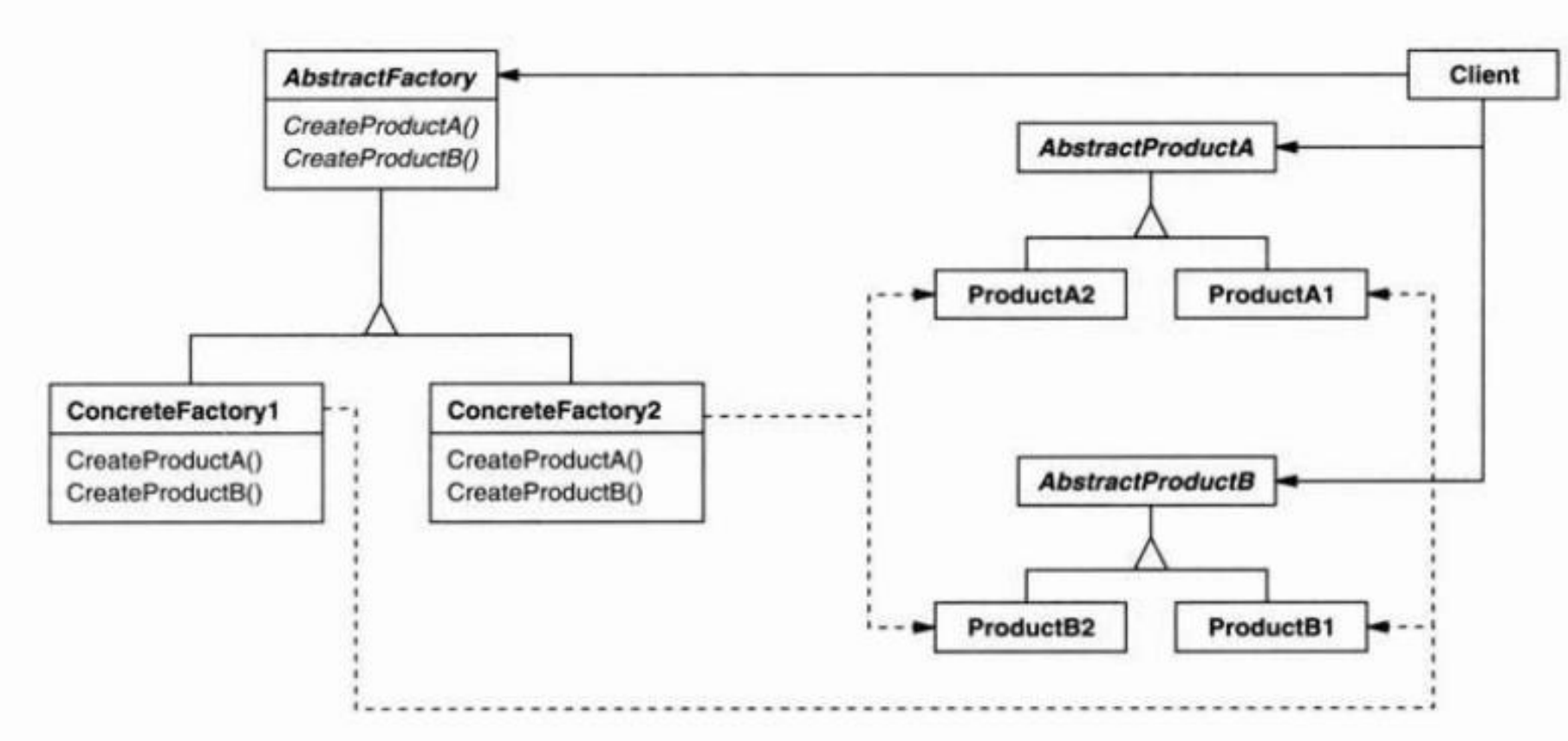
Solution - Factory



AbstractFactory (Gof)



AbstractFactory (Gof)



Question:

- Qui devrait créer la **Factory**, considérant que nous n'avons besoin que d'une seule instance accessible de partout dans notre logiciel?

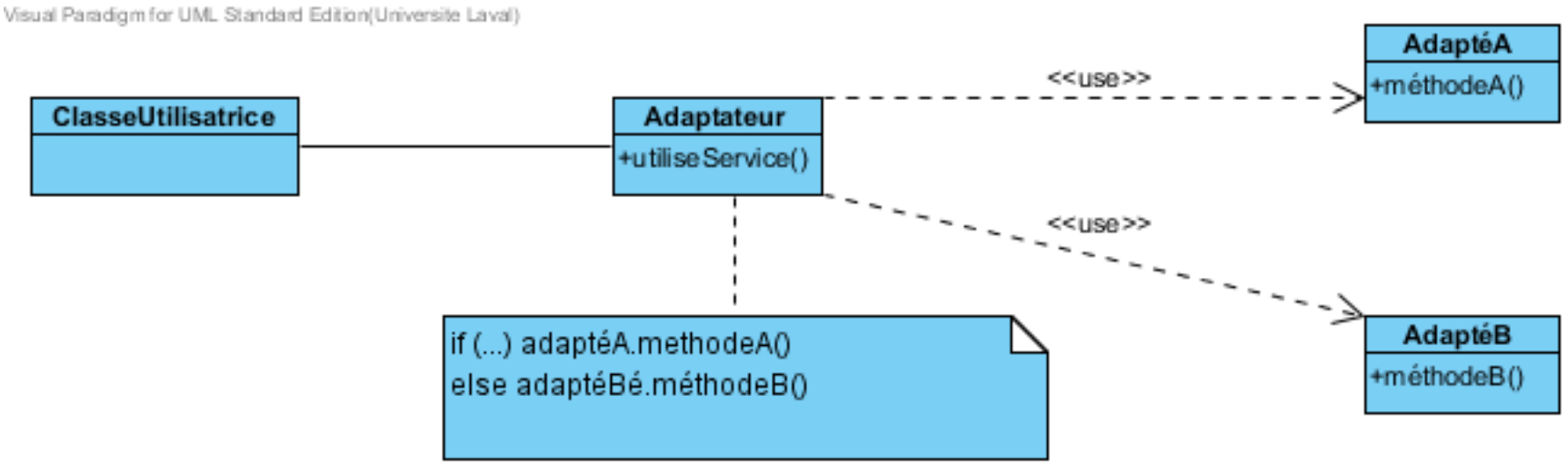
Question:

- Qui devrait créer la **Factory**, considérant que nous n'avons besoin que d'une seule instance accessible de partout dans notre logiciel?
- **Option 1**: créer une variable globale
- **Option 2**: passer une référence sur cette instance à toutes les classes qui en ont besoin
- **Option 3**: appliquer le patron « Singleton »

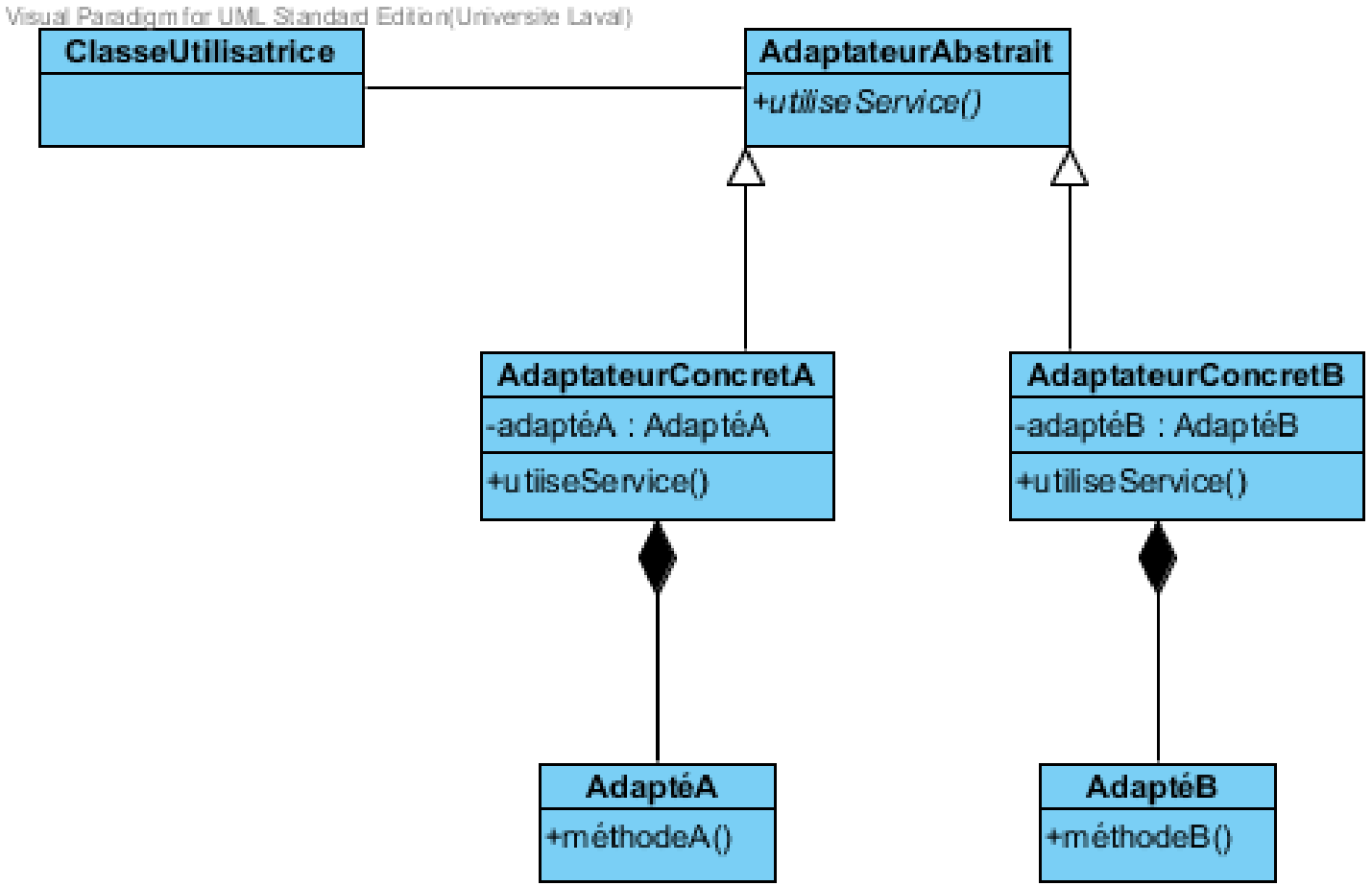
Adapter (GoF)

- **Problème**: Utilisation souhaitée de plusieurs composants alternatifs qui présentent des « interfaces » différentes
- **Solution**: Mettre un objet intermédiaire entre les deux (l'adaptateur) qui nous sert d'interprète

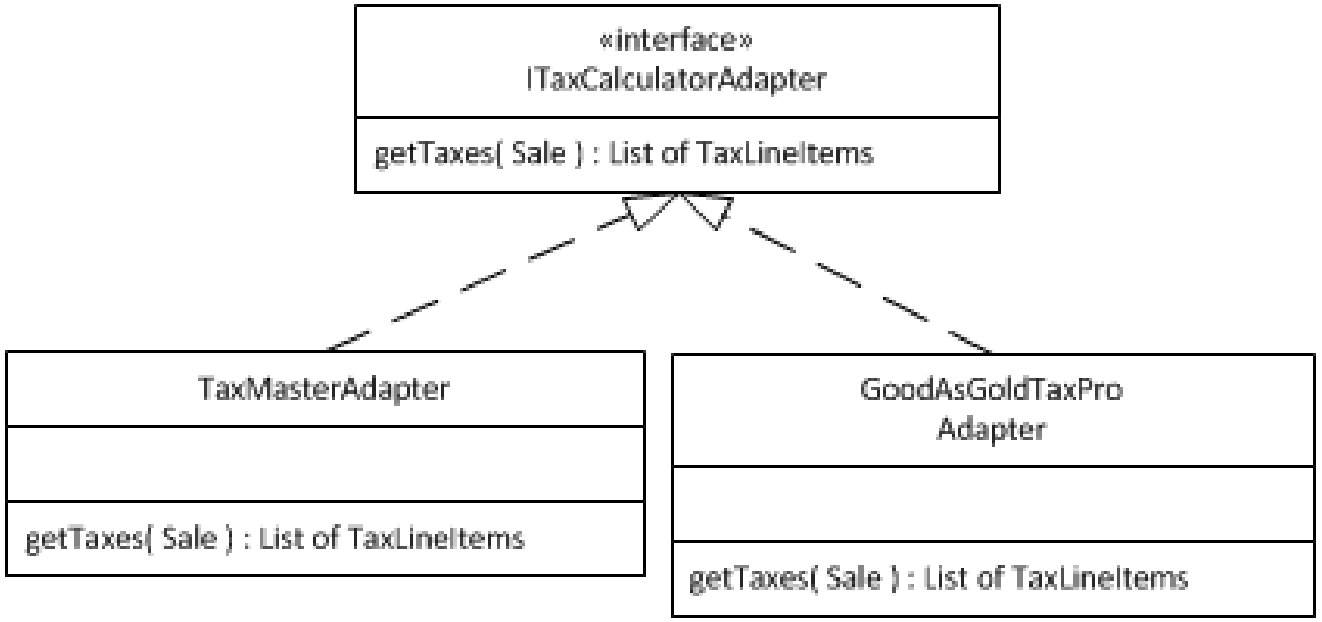
Méthode de base



Méthode proposée par GoF

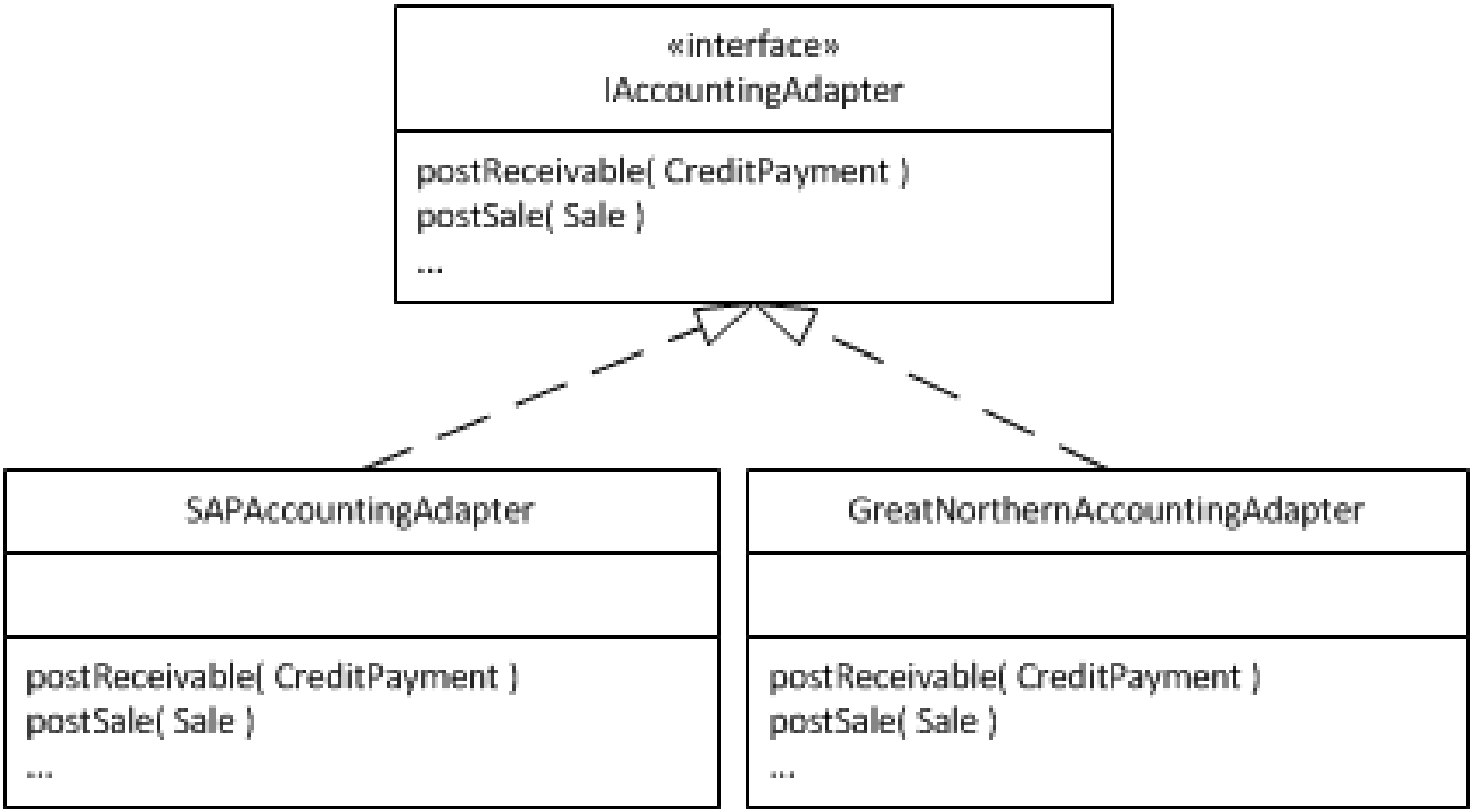


Application NexGen POS

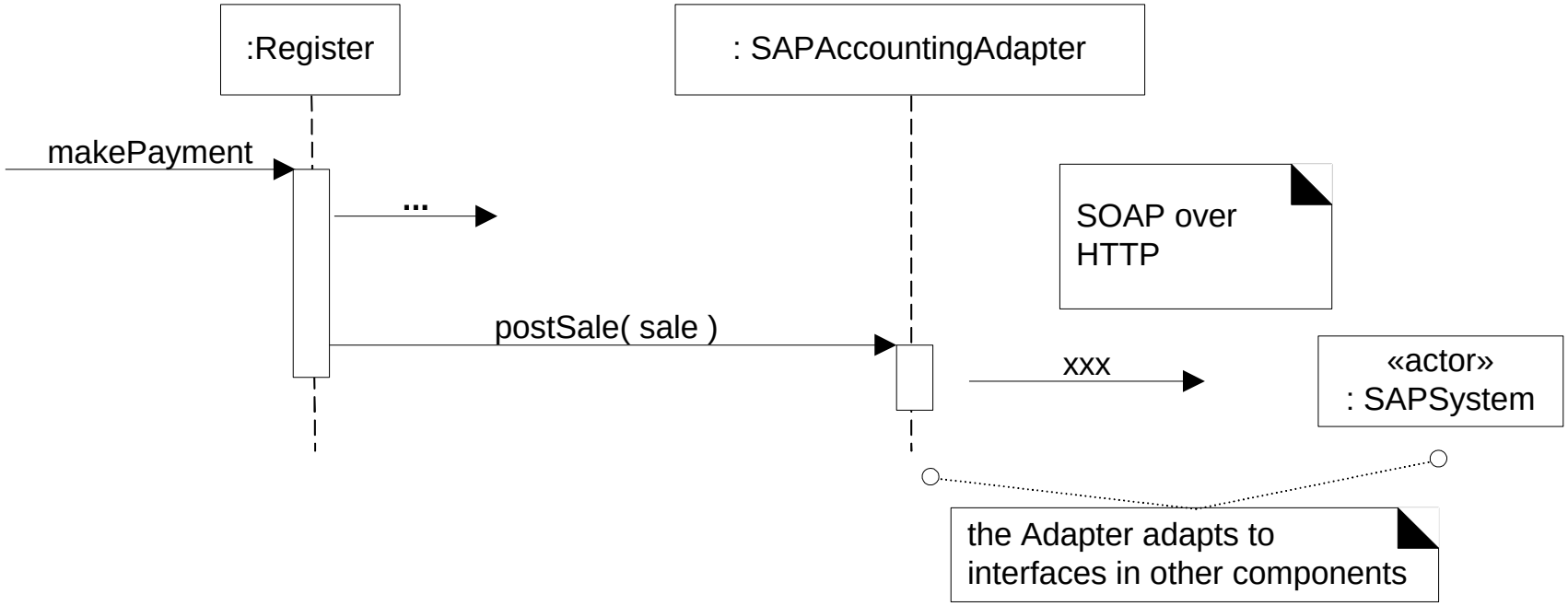


Adapters use interfaces and polymorphism to add a level of indirection to varying APIs in other components.

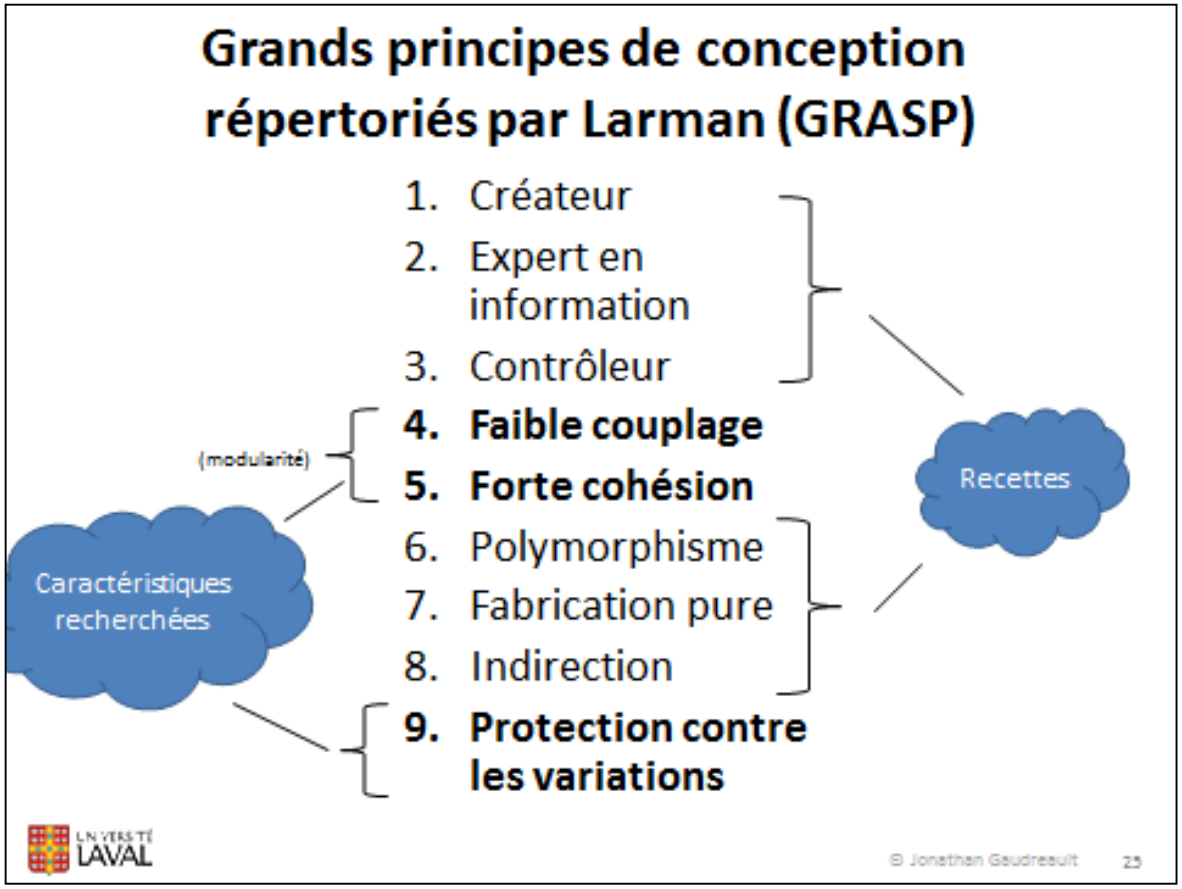
Application NexGen POS



Application NexGen POS

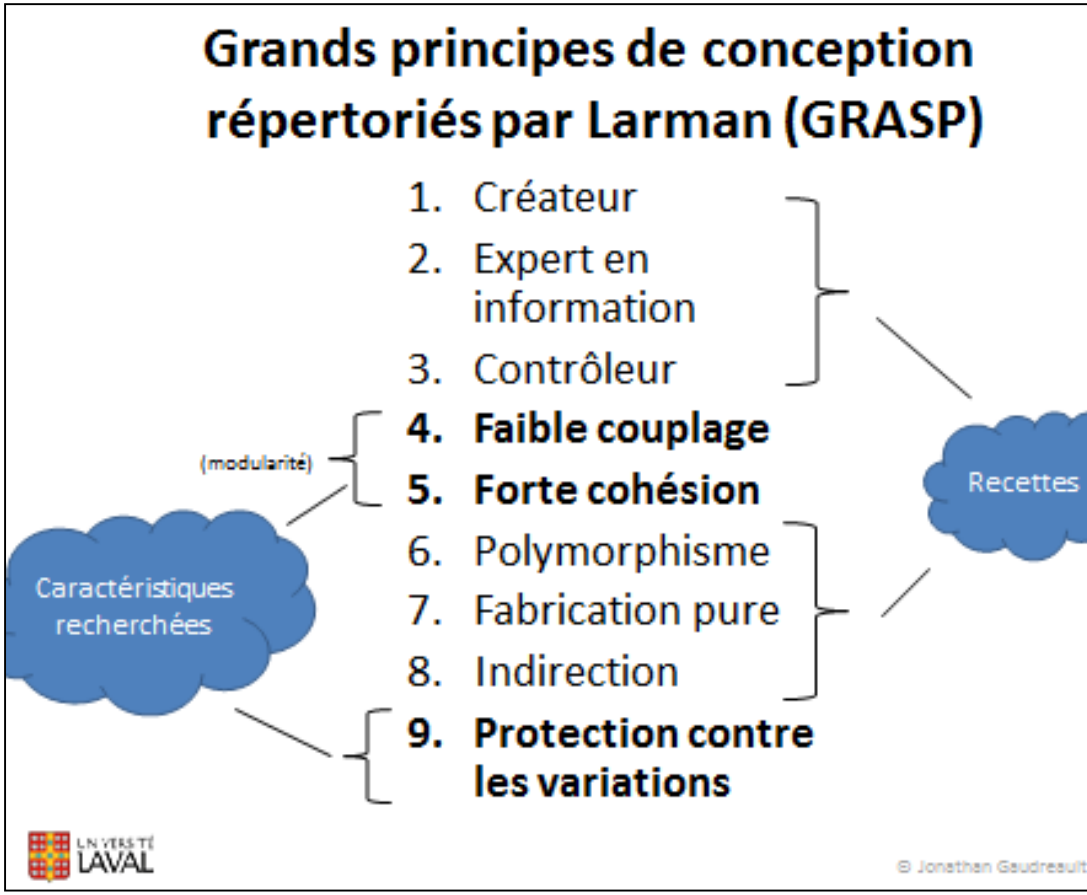


Grands principes GRASP qu'on y retrouve?



Adaptateur: Grands principes GRASP qu'on y retrouve?

- **Caractéristiques recherchées**:
  - Protection contre les variations
  - Faible couplage
  - Forte cohésion (?)
- **Recettes**:
  - Polymorphisme
  - Indirection
  - Fabrication pure (?)

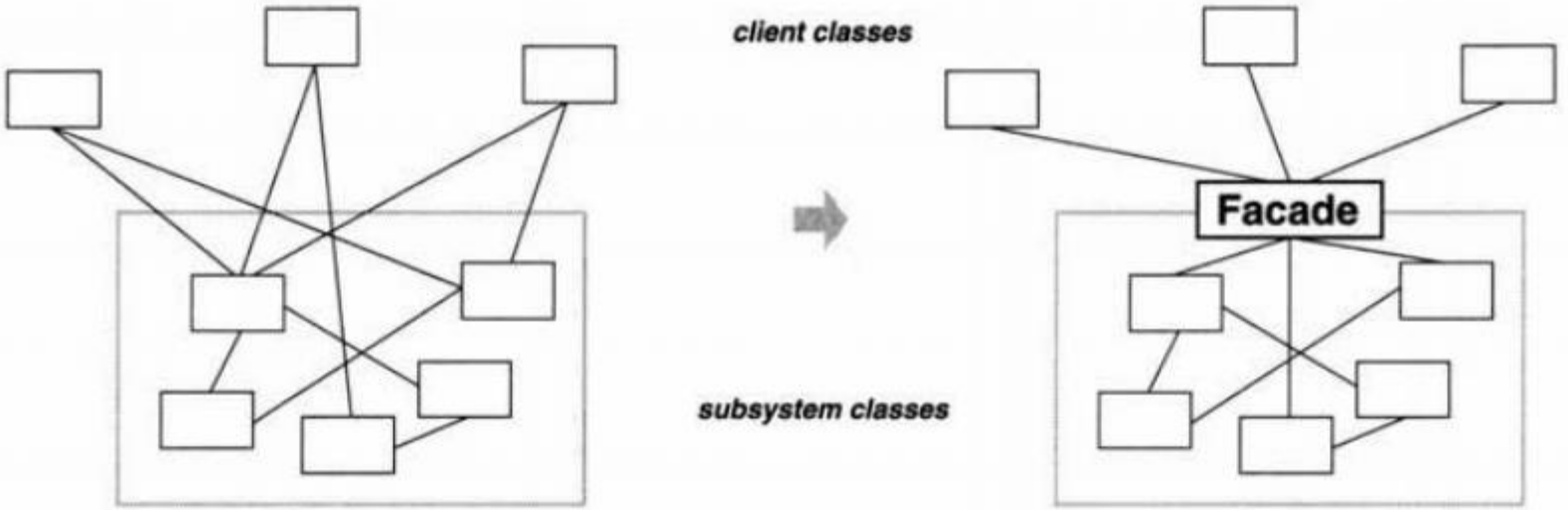


Façade (GoF)

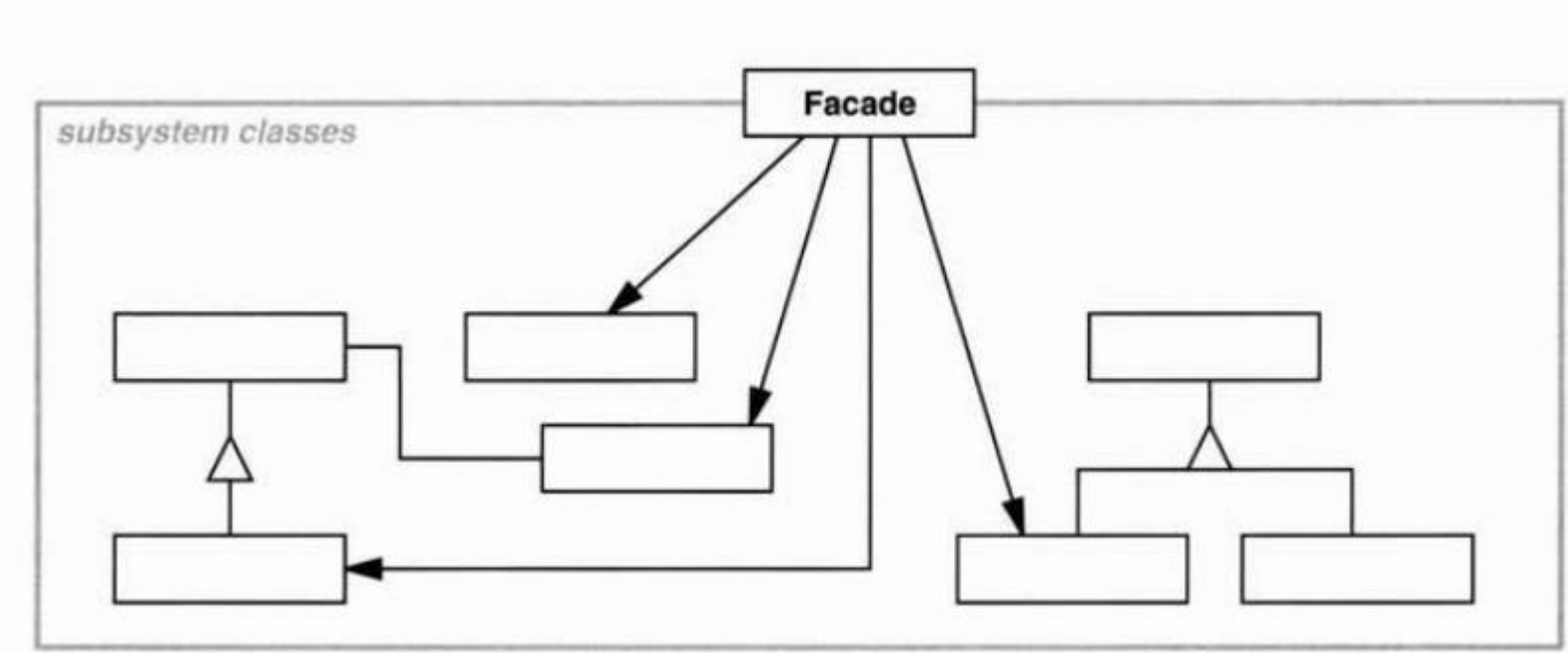
- **Problème**: On désire définir un point de contact unique pour plusieurs objets disparates constituant en quelque sorte un sous-système
- **Solution**: Définir une « façade » qui encapsule ce sous-système. Il présentera une interface unifiée de plus haut niveau rendant le sous-système plus facile à utiliser.

[Gamma et al]

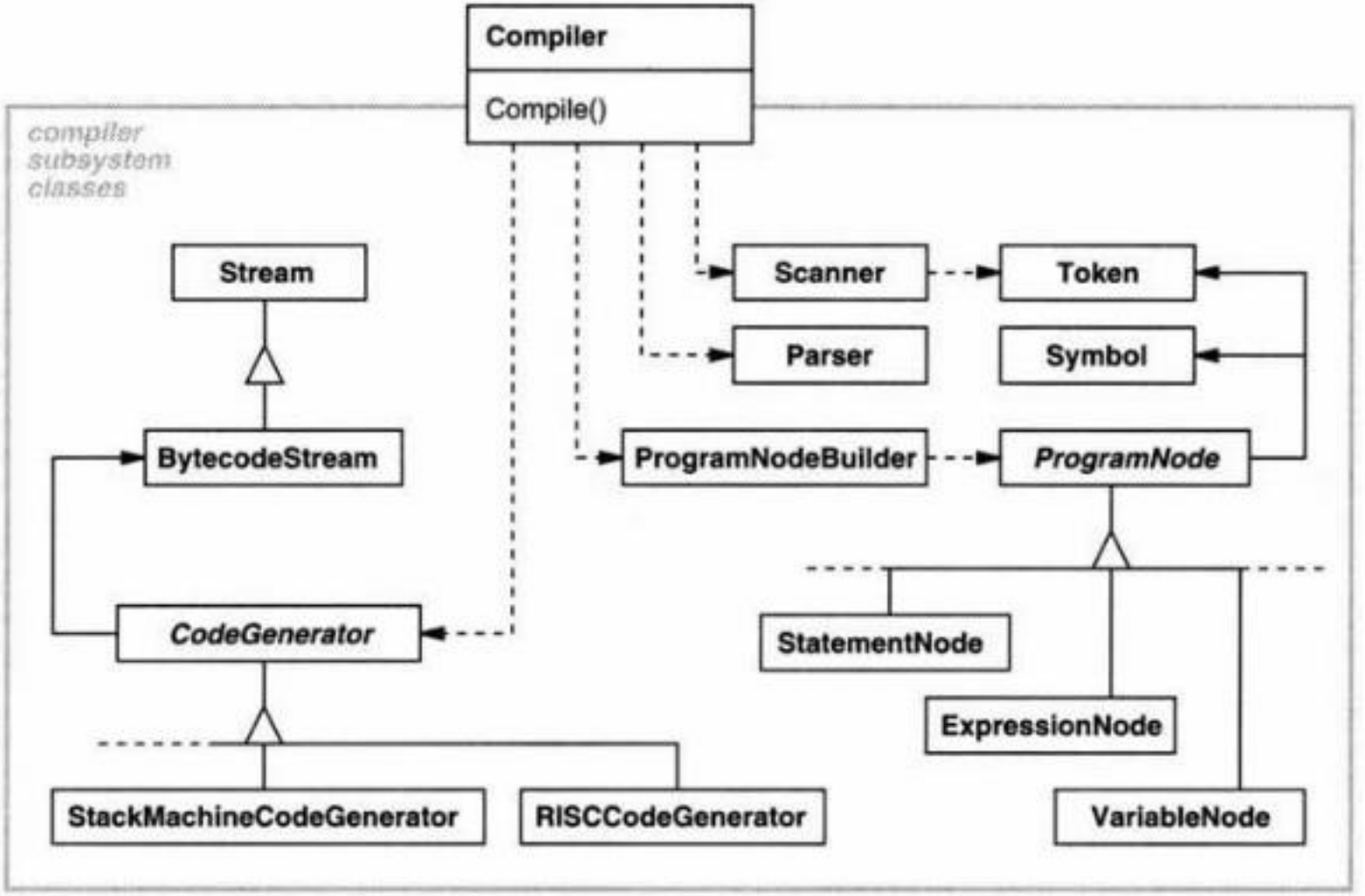
Structuring a system into subsystems helps reduce complexity. A common design goal is to minimize the communication and dependencies between subsystems. One way to achieve this goal is to introduce a **facade** object that provides a single, simplified interface to the more general facilities of a subsystem.



- Les classes utilisatrices (« clients ») communiquent avec le sous-système via l'objet Facade, qui s'occupe de rediriger vers les bons objets.



Exemple de façade - Compilateur



Avantages

- Le client travaille avec un moins grand nombre d'objets (le sous-système est plus facile à utiliser)
- La complexité interne est cachée
- On a un couplage faible; on peut modifier l'intérieur du sous système sans avoir d'impact à l'extérieur (protection contre les variations)

Et dans votre projet?

Et dans votre projet?

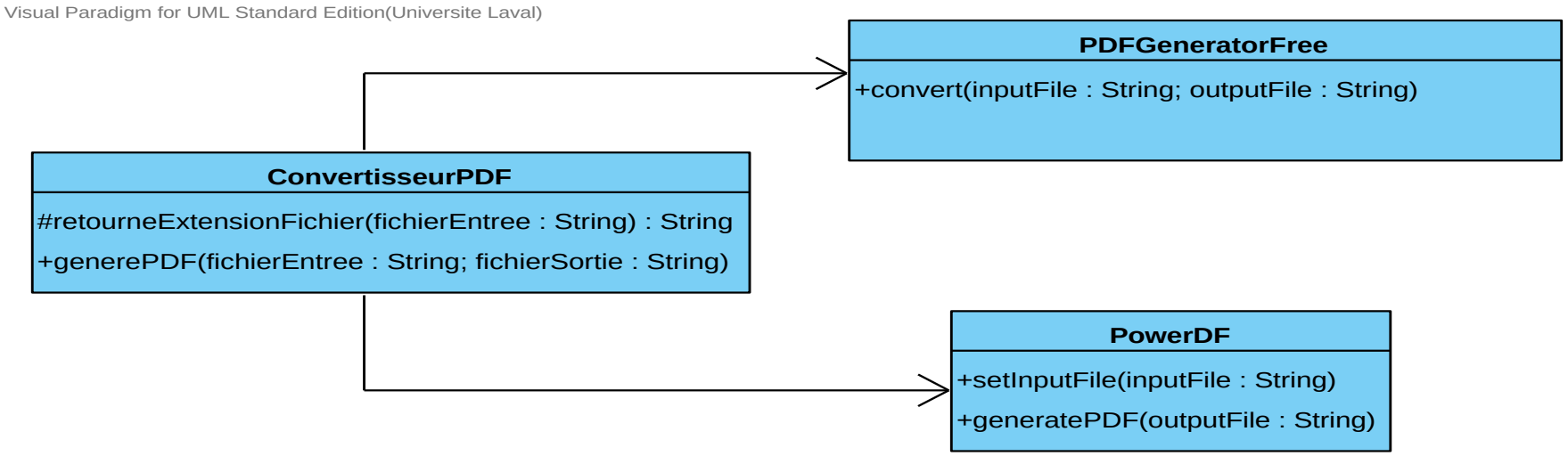
- Contrôleur!

Exercice

- Votre entreprise (une grande compagnie d'assurance) reçoit des réclamations d'assurance accompagnées de fichiers électroniques de type Word (extension « .docx ») et Excel (extension « .xlsx »).
- Pour réduire le volume de donnée sur le serveur on propose de les convertir en PDF au fur et à mesure qu'elles sont reçues par le serveur.
- Vous avez trouvé deux convertisseurs open-source :
  - Une classe appelée **PDFGeneratorFree** qui offre une méthode « convert(inputFile : String, outputFile : String) » qui prends en paramètre le chemin du fichier d'entrée et le chemin du fichier de sortie. Cette classe supporte la conversion des documents word seulement.
  - Une appelée **PowerDF** qui supporte la conversion des documents Excel seulement. Pour faire la conversion on doit appeler successivement deux méthodes : setInputFile(inputFile : String) puis generatePDF(outputFile : String).
- Créer une classe **ConvertisseurPDF** qui constitue une facade pour accéder aux deux autres classes (en éventuellement à d'autres convertisseur dans le futur). Cette classe comporterait deux méthodes.
  - L'une est publique : « genererePDF(fichierEntree : String; fichierSortie : String) ».
  - L'autre est protégée : « retourneExtensionFichier(fichierEntree : String) : String ».

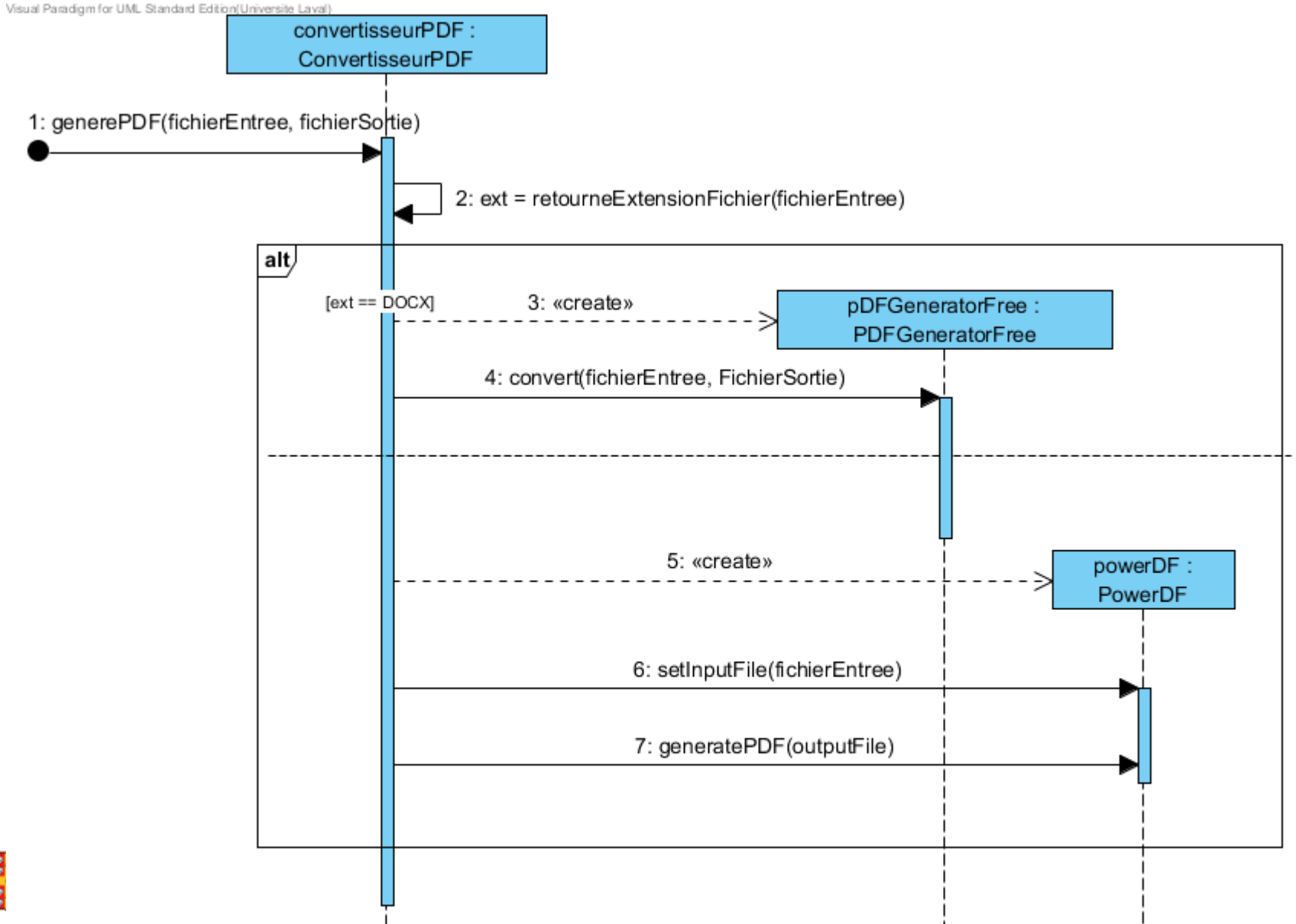
Solution - Créez le diagramme de classe

Solution - Créez le diagramme de classe



Solution - Créez le diagramme de séquence

Solution - Créez le diagramme de séquence



Adaptateur ou Façade ?

Adapter	Façade
Besoin d'utiliser une classe existant déjà, mais son interface n'est pas celle qu'on désire utiliser.	Lorsqu'on désire unifier et/ou simplifier l'accès à un sous-système complexe (souvent constitué de plusieurs objets.
L'adaptateur transforme l'interface de l'objet en une autre convenant mieux à l'objet utilisateur.	La façade découple le lien entre la classe utilisatrice et le sous-système en question.
L'implémentation peut être très simple: on ne fait que rediriger des appels.	Chaque appel à la façade requière potentiellement l'appel à plusieurs classes et méthodes de celles-ci.

À faire pour ce module

- Lecture des chapitres
  - Version anglaise: 25, 26 (en mettant l'accent sur les patrons étudiés dans le cours)
  - Version française: 22, 23 (en mettant l'accent sur les patrons étudiés dans le cours)
- Comprendre les patrons présentés
- Comprendre la relation avec les grands principes de design orienté objet (GRASP)