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

Automne 2016

Module 17 - Patrons de conception (partie 3)

Martin.Savoie@ift.ulaval.ca

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

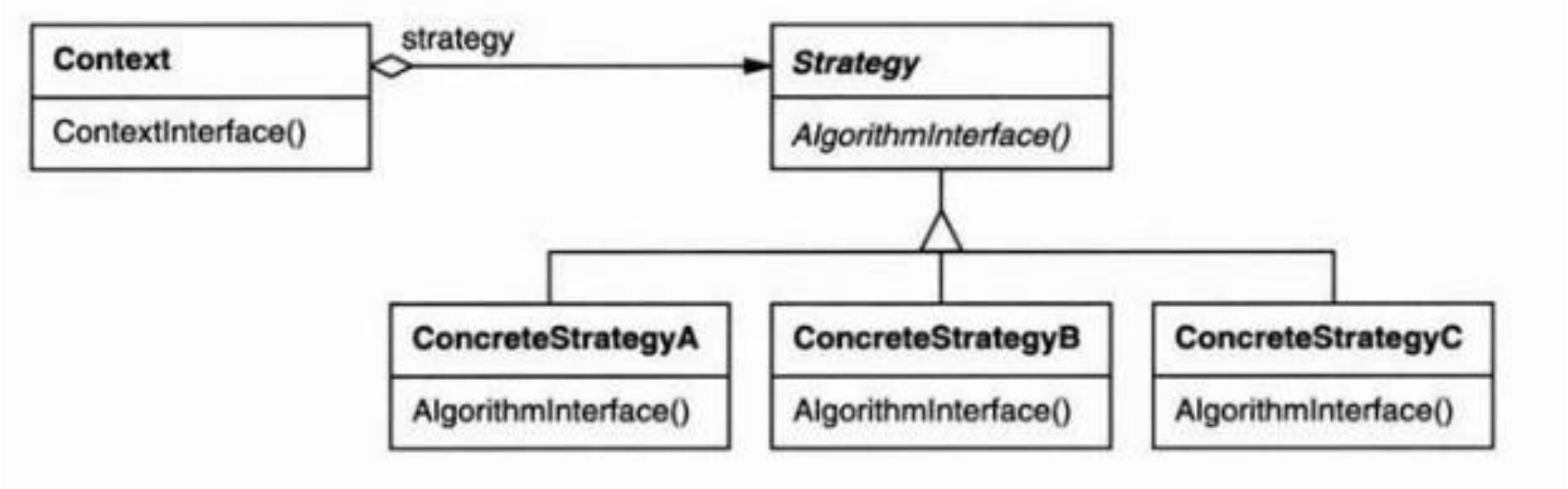
UNIVERSITÉ LAVAL

Strategy (GoF)

- **Problème**: Différentes versions d'un algorithme (ou d'une politique, ou d'une *stratégie*) sont appelés à coexister.
- **Solution**: Définir les versions de l'algorithme comme des classes distinctes (possédant une interface commune), plutôt qu'en tant que méthode…

  … la méthode devient une classe!

Strategy



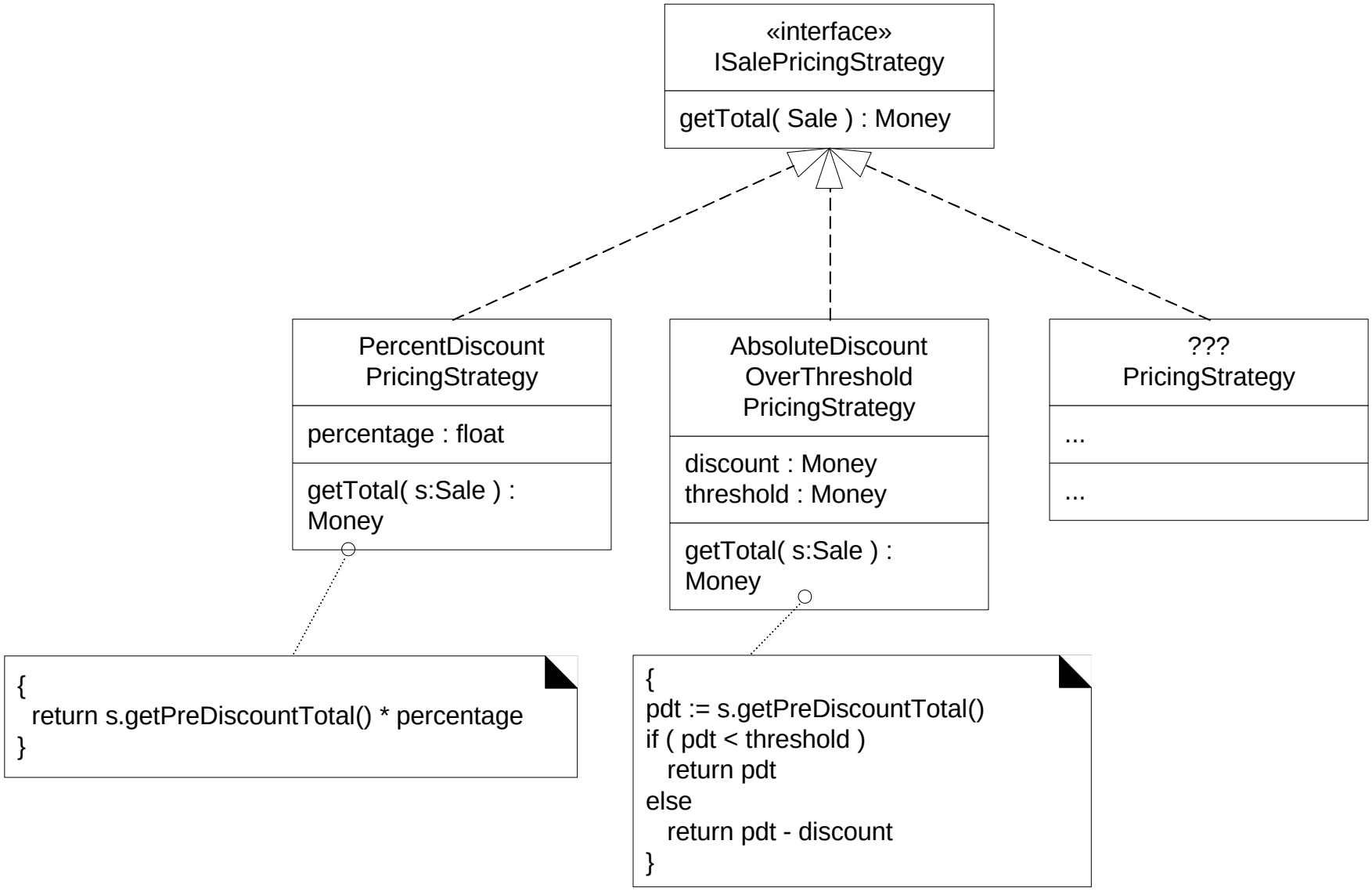
Quel est l'avantage d'encapsuler l'algorithme dans une classe plutôt que dans une méthode de la classe utilisatrice?

Quel est l'avantage d'encapsuler l'algorithme dans une classe plutôt que dans une méthode de la classe utilisatrice?

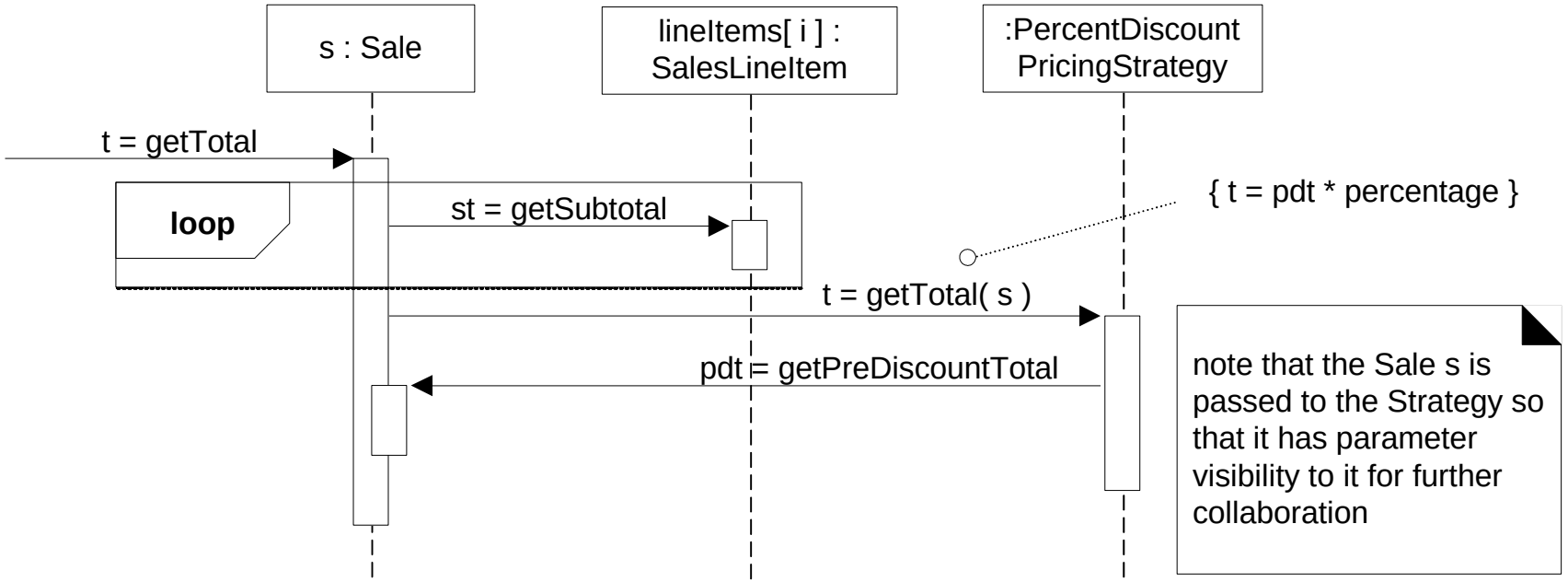
Quel est l'avantage d'encapsuler l'algorithme dans une classe plutôt que dans une méthode de la classe utilisatrice?

- Le choix de l'algorithme à utiliser n'est pas « hard-codé » dans la classe utilisatrice (on peut passer à l'objet utilisateur une référence sur l'algorithme à utiliser)
  - Le choix est fait au « runtime » plutôt qu'au moment de la compilation
- Les différentes versions de l'algorithme peuvent être utilisées/partagées par plusieurs classes
- La hiérarchie de la famille d'algorithmes évolue indépendamment de la hiérarchie des classes utilisatrices

Exemple NexGen – Calcul des rabais



Exemple NexGen – Calcul des rabais



- Sauriez-vous écrire le code correspondant à ce diagramme?

Détail intéressant dans ce design

- On passe à la méthode getTotal() l'objet « Sale » plutôt que le total avant rabais.

- Quels en sont les avantages?

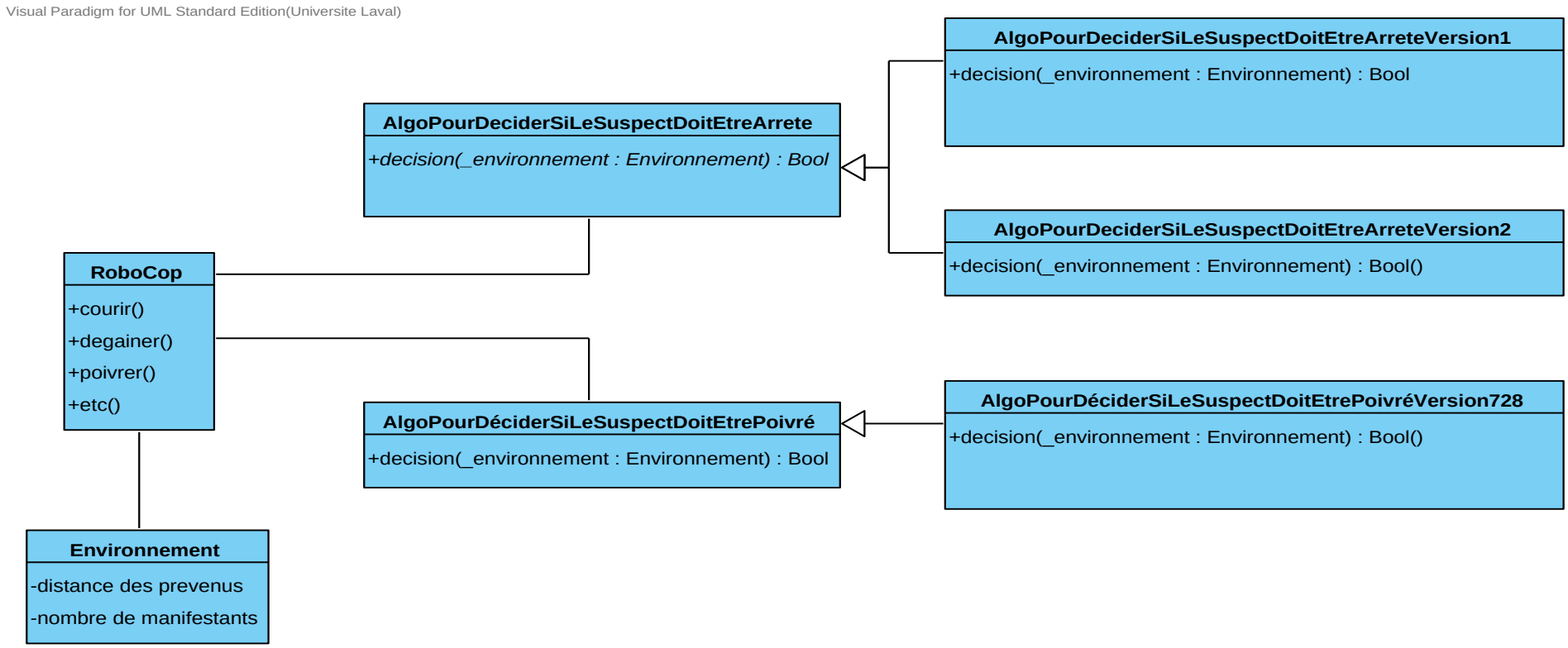
Détail intéressant dans ce design

- On passe à la méthode getTotal() l'objet « Sale » plutôt que le total avant rabais.

- Quels en sont les avantages?

  – Davantage de possibilités au niveau de ce qu'un rabais peut faire (ex: 3 articles pour le prix de 2)

RoboCop


Visual Paradigm for UML Standard Edition(Universite Laval)
AlgoPourDeciderSiLeSuspectDoitEtreArreteVersion1
+decision(_environnement : Environnement) : Bool
AlgoPourDeciderSiLeSuspectDoitEtreArrete
+decision(_environnement : Environnement) : Bool
AlgoPourDeciderSiLeSuspectDoitEtreArreteVersion2
+decision(_environnement : Environnement) : Bool()
RoboCop
+courir()
+degainer()
+poivrer()
+etc()
AlgoPourDéciderSiLeSuspectDoitEtrePoivré
+decision(_environnement : Environnement) : Bool
AlgoPourDéciderSiLeSuspectDoitEtrePoivréVersion728
+decision(_environnement : Environnement) : Bool()
Environnement
-distance des prevenus
-nombre de manifestants

Réflexion

- Quel est l'avantage de cette approche plutôt que simplement créer plusieurs sous-classes de RoboCop, chacune ayant le comportement désiré?

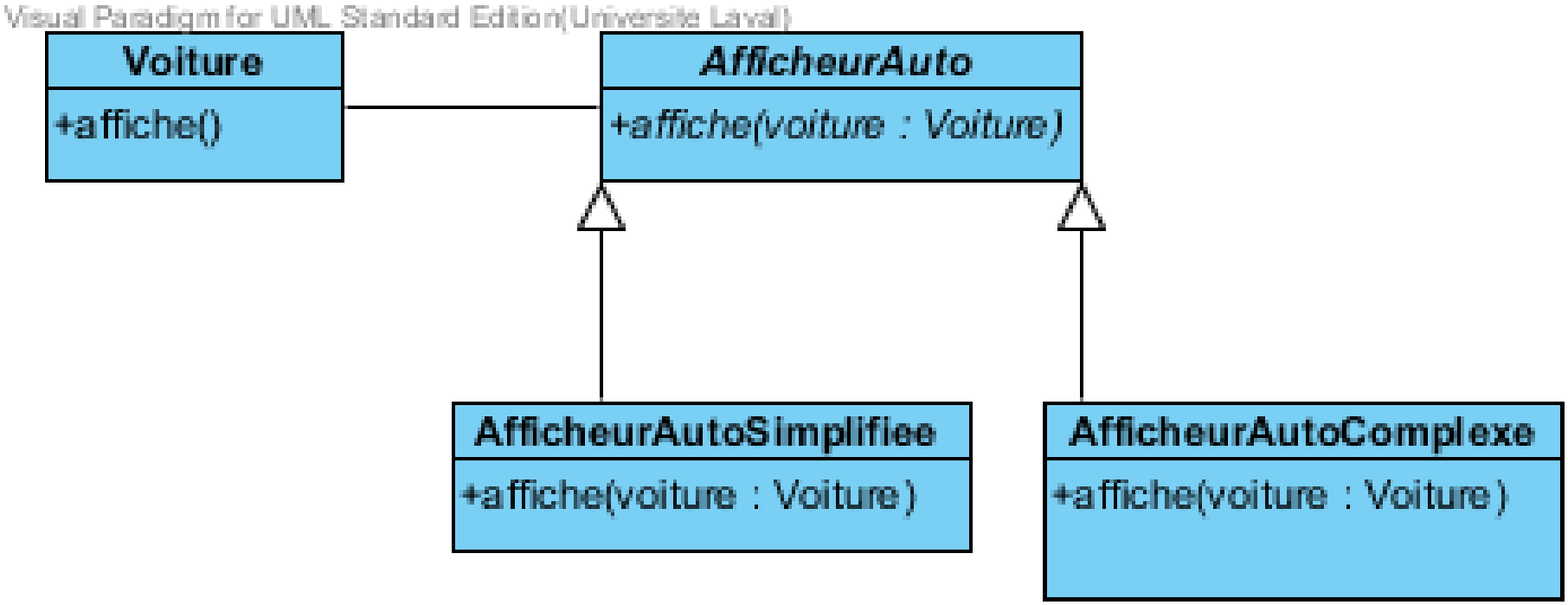
- Dans une vraie application, qui assignerait les stratégies au RoboCop?

Exercice - Strategy

- Vous travaillez au développement d'un logiciel permettant de concevoir et dessiner des voitures en trois dimensions. Une classe appelée **Voiture** possède une méthode appelée *affiche*(…) qui permet d'afficher la voiture à l'écran.
- Le logiciel doit permettre de l'afficher de manière détaillée, ou bien de manière simplifiée (plus rapide mais moins beau).
- Exploitez le patron « Strategy » pour permettre à la classe Voiture de se dessiner à l'écran avec l'opération *affiche*() en utilisant l'une de deux options (choisie dynamiquement au moment de l'exécution) qui seront implantées par deux classes différentes: **AfficheurAutoSimplifiee** et **AfficheurAutoComplexe**.
- Tracez un diagramme de classe
- Tracez un diagramme de séquence

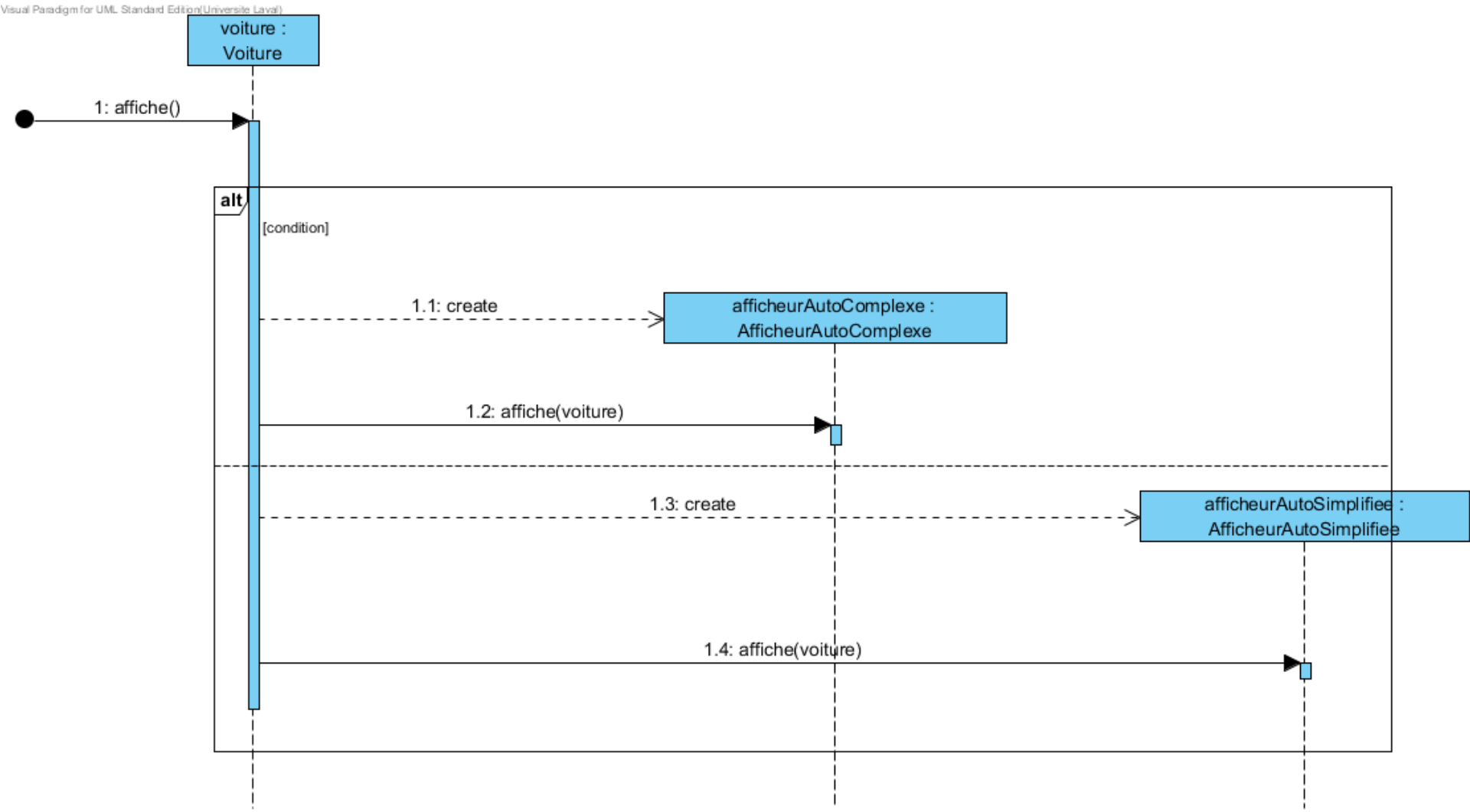
Solution - Diagramme de classes

Solution - Diagramme de classes



Solution - Diagramme de séquence

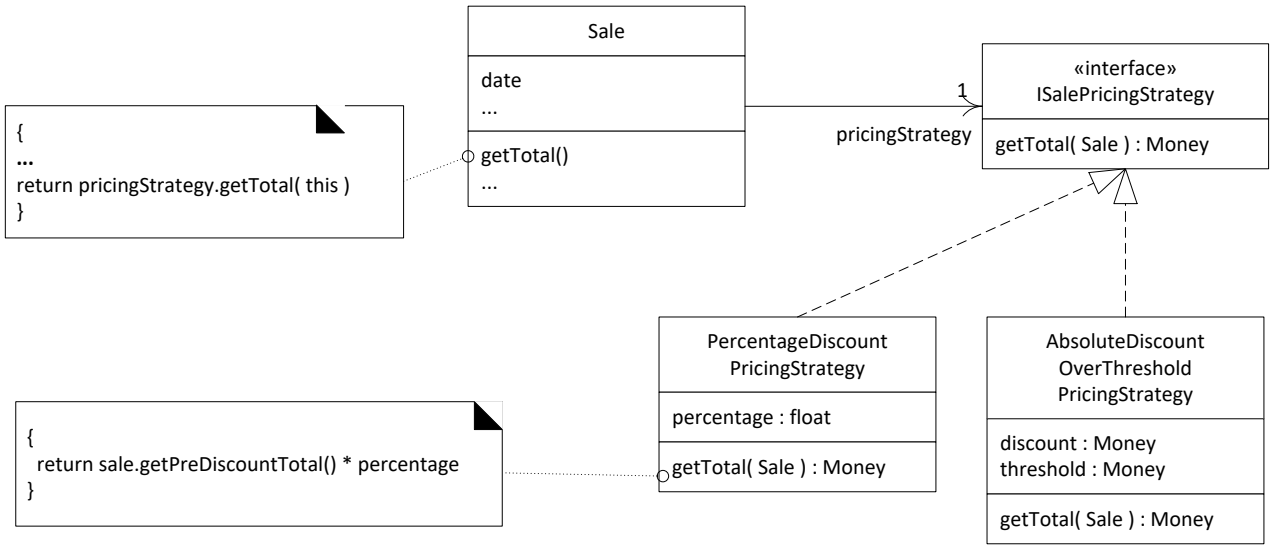
Solution - Diagramme de séquence



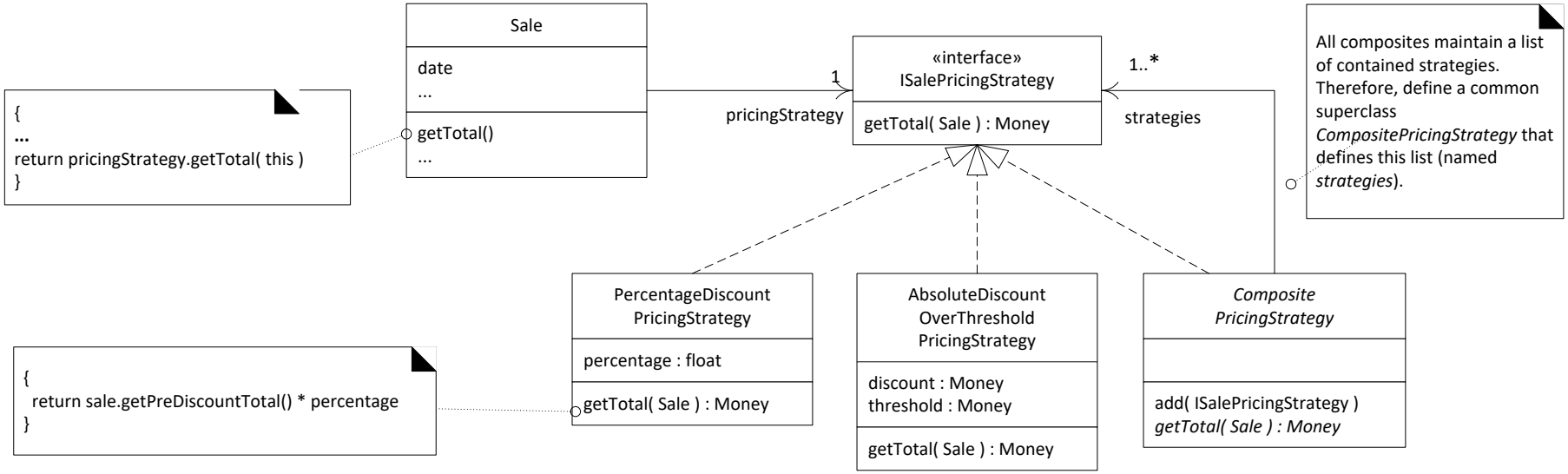
Composite (GoF)

- **Problème**: Comment traiter un groupe d'objet (une composition) de la même manière (polymorphique) qu'un des objets individuel?
- **Solution**: S'assurer que le « conteneur » implémente la même interface que le « contenu »

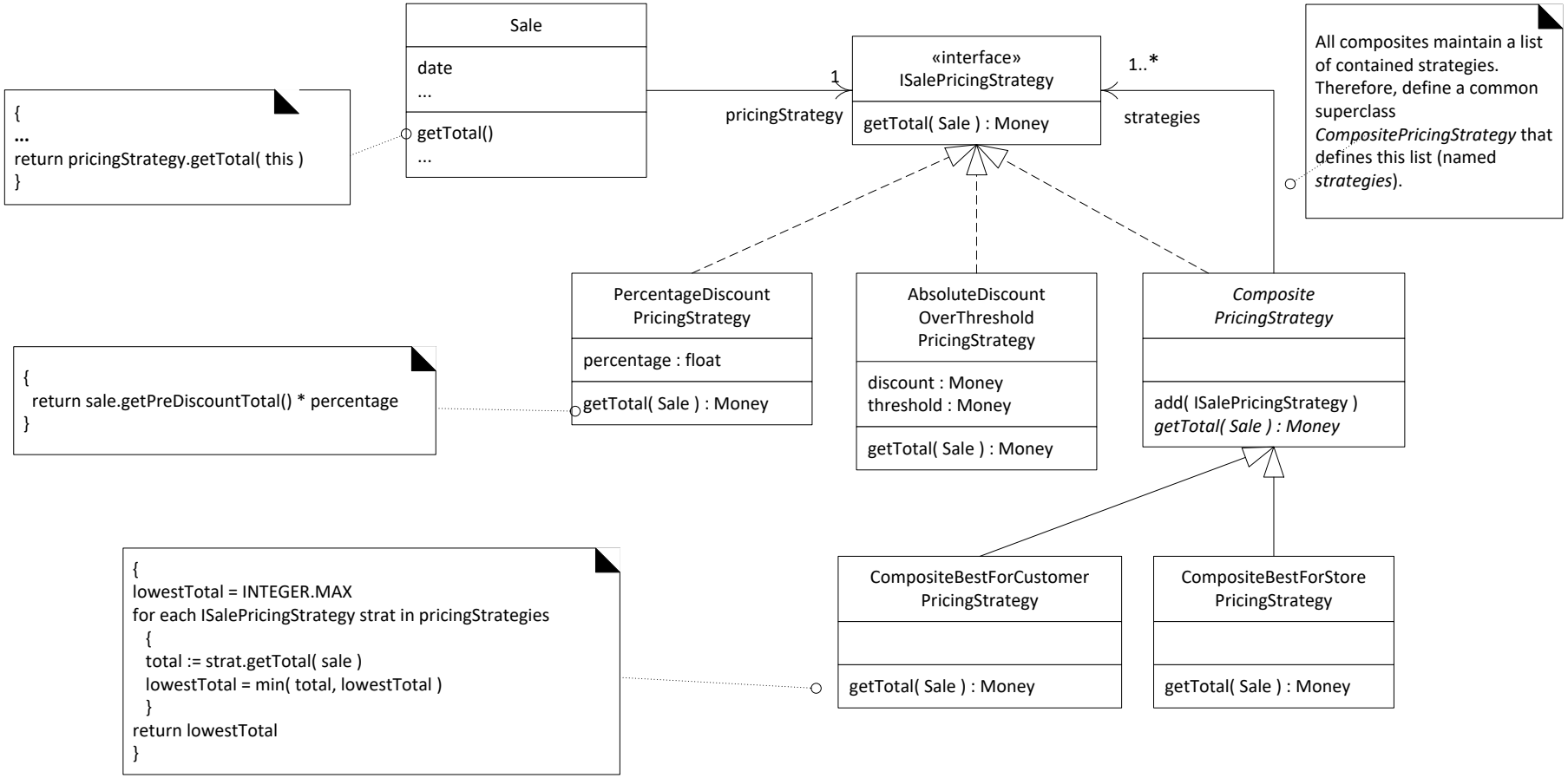
Exemples NexGen – Un rabais qui est une combinaison de rabais



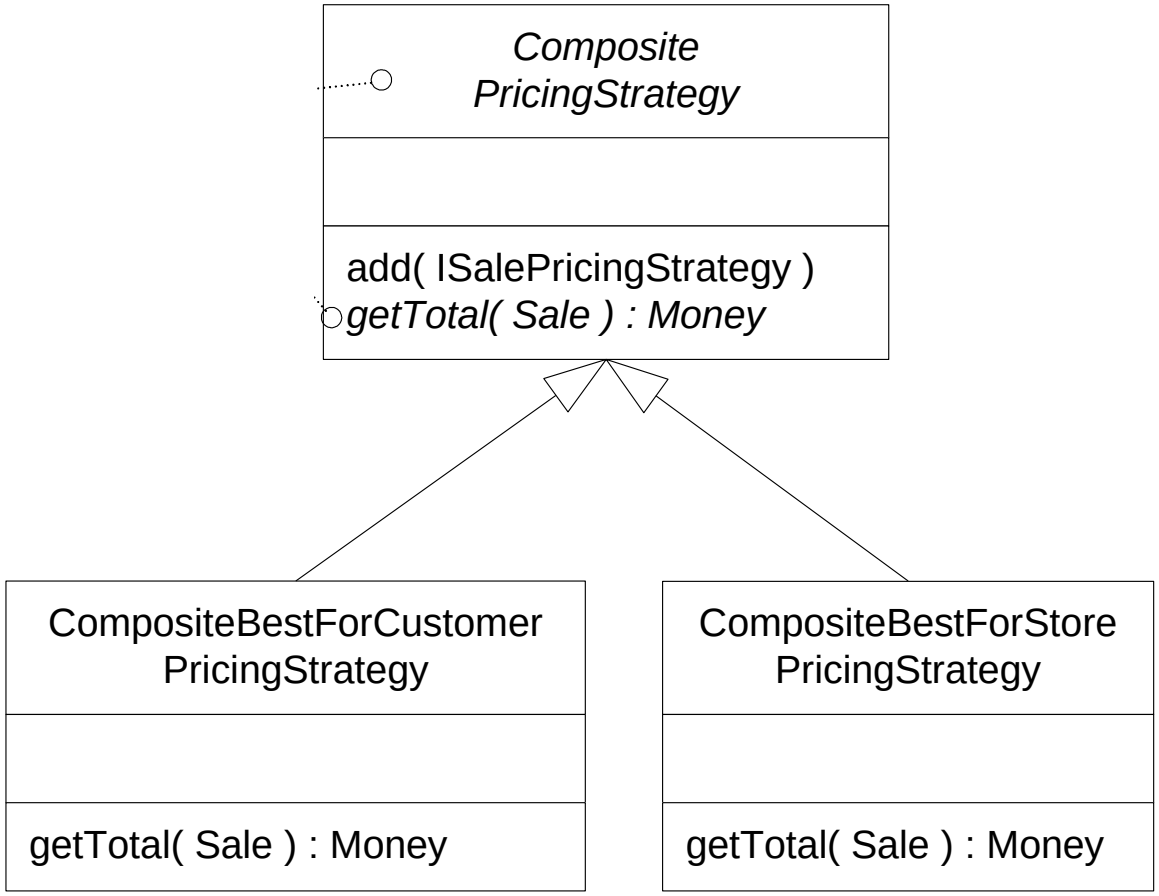
Exemples NexGen – Un rabais qui est une combinaison de rabais



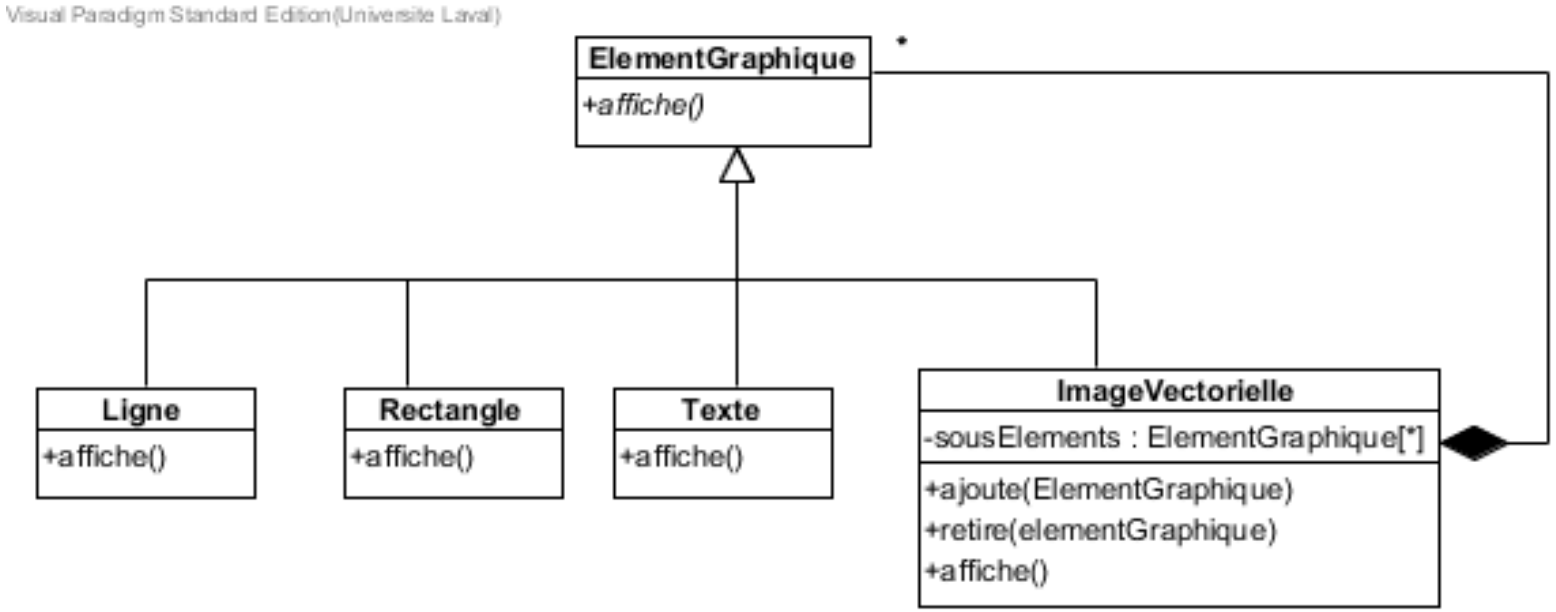
Exemples NexGen – Un rabais qui est une combinaison de rabais



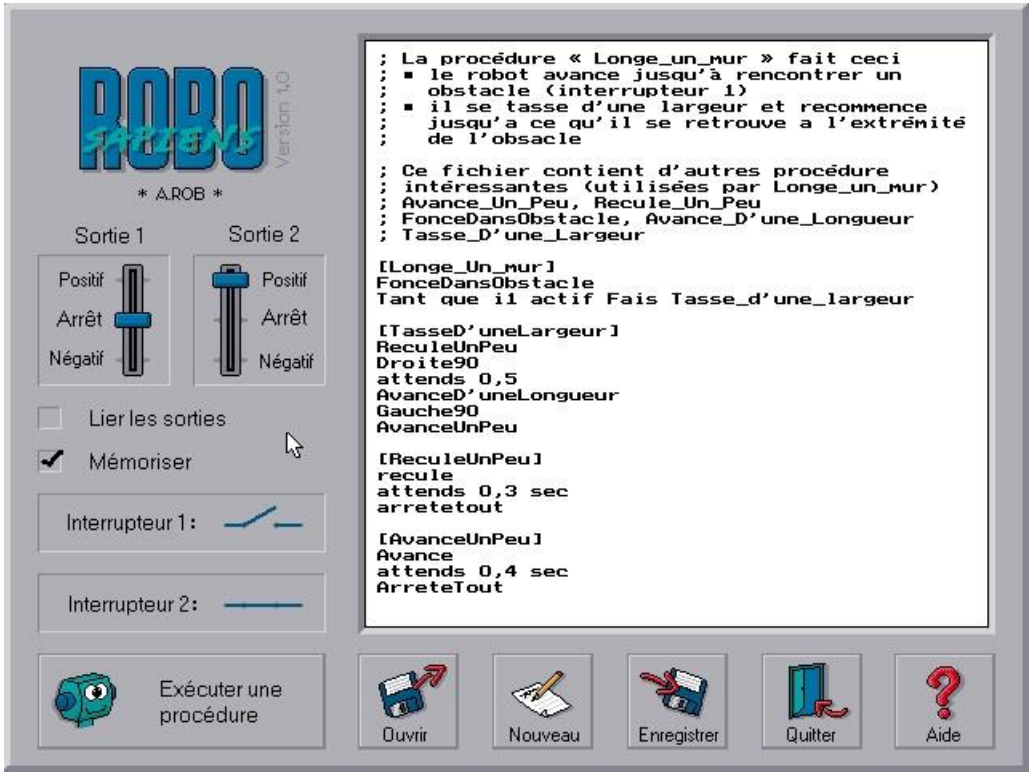
NexGenPos – Un rabais peut être une combinaison de rabais



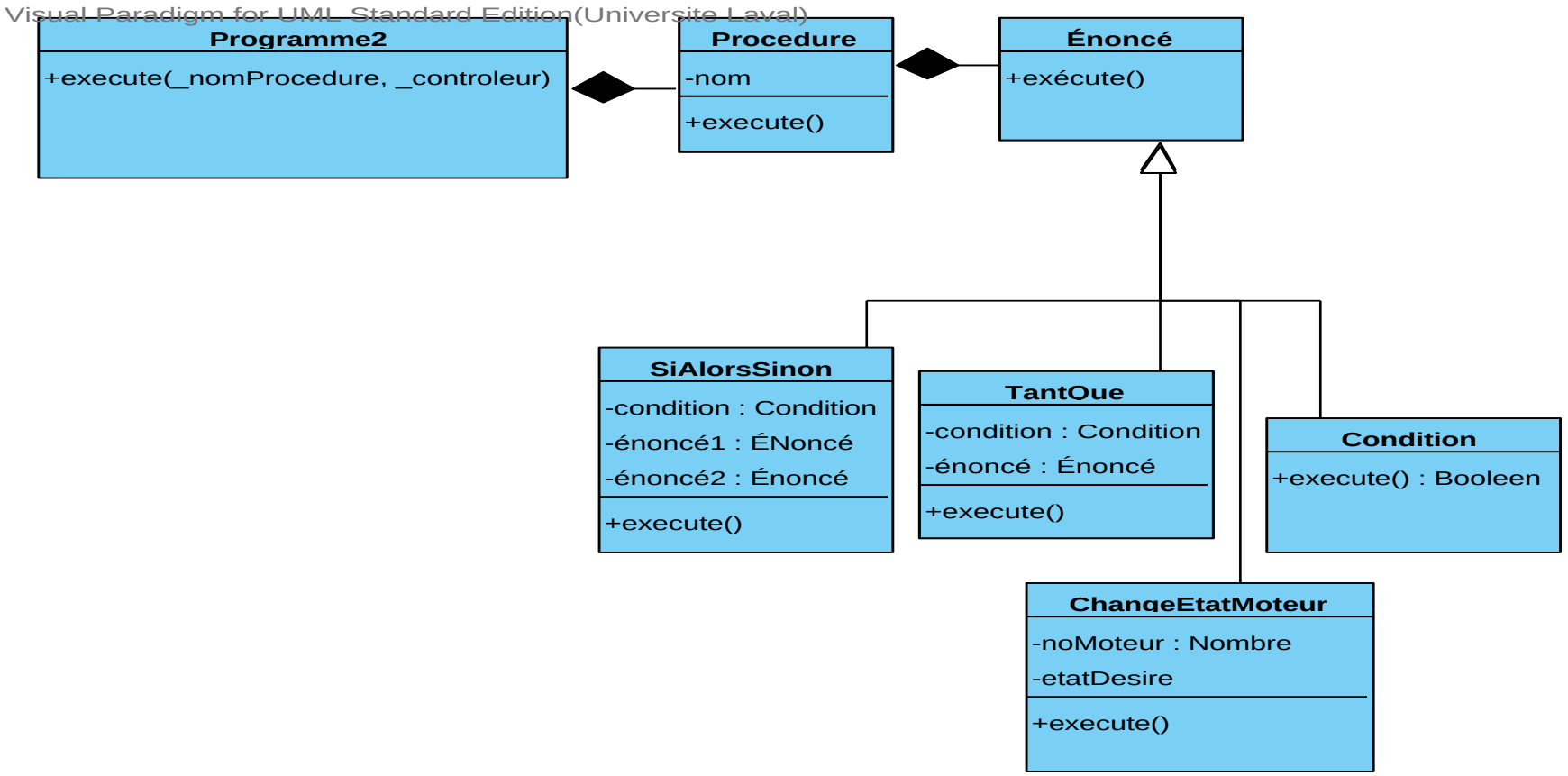
Système graphique



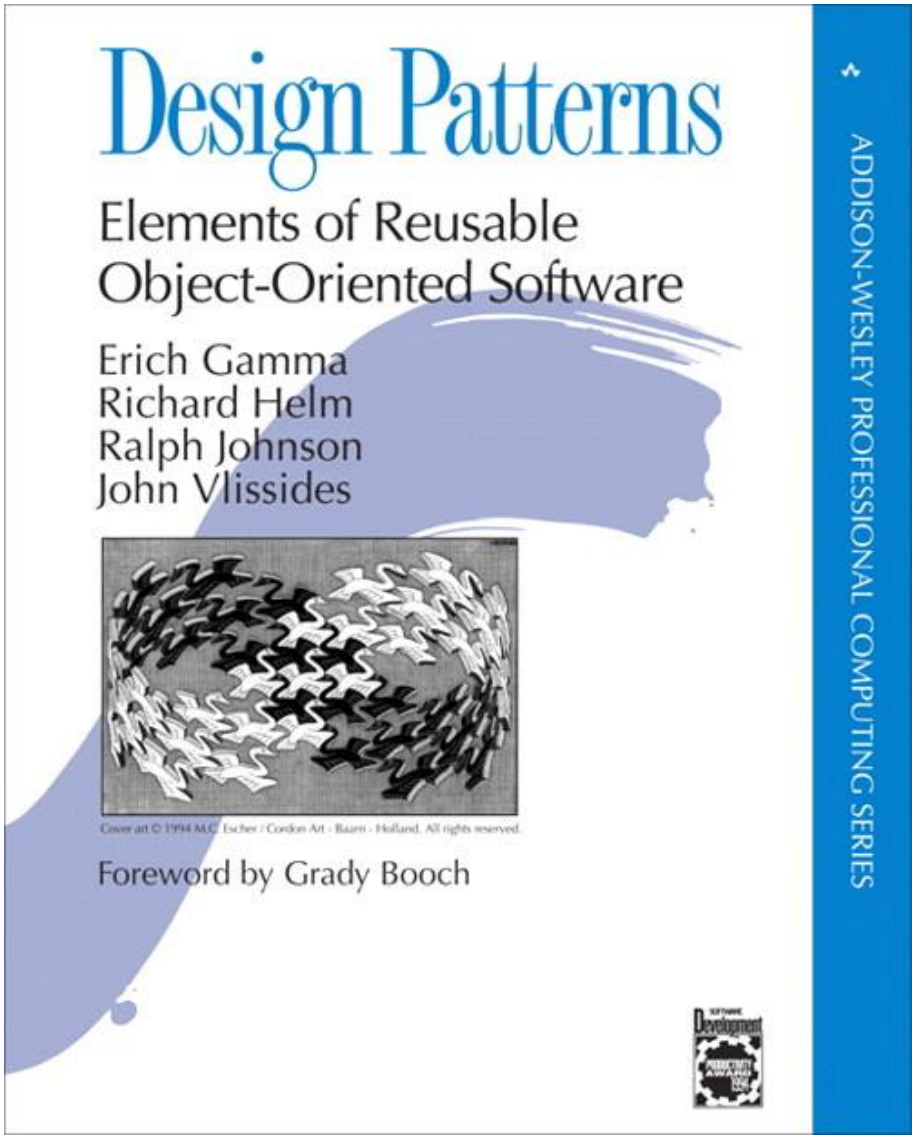
Nous avons vu une autre application de ce principe plus tôt dans la session…



Exemple Robo sapiens



Étude de cas


Design Patterns
Elements of Reusable
Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

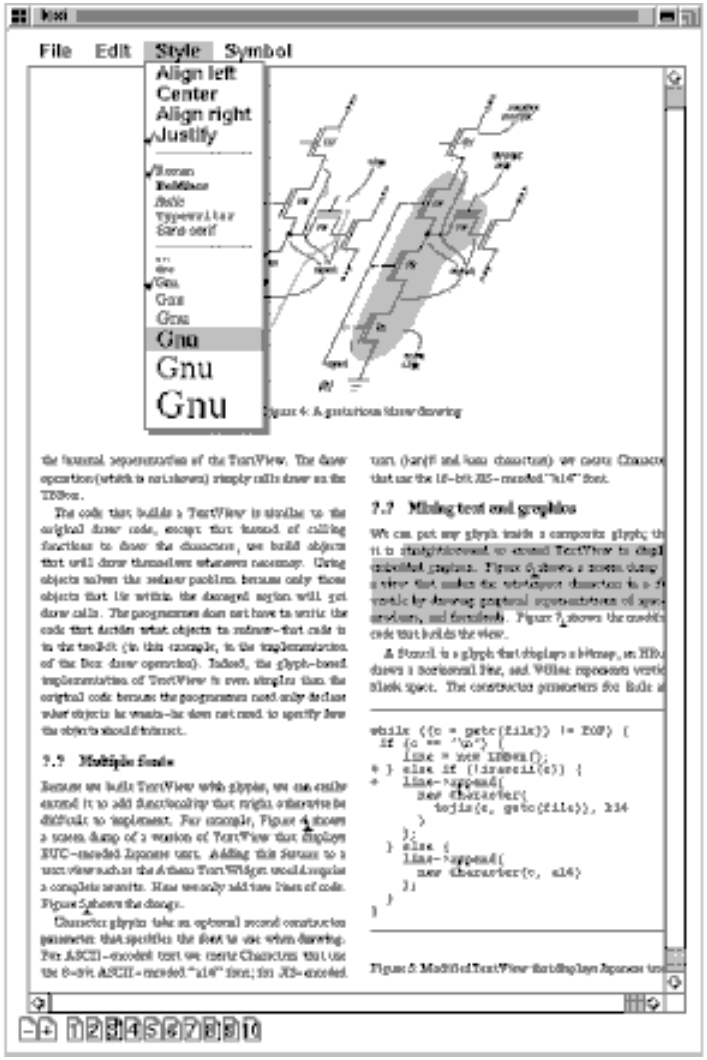
Design Patterns : Elements of Reusable Object-Oriented Software [Gamma et al]

1. Introduction
2. Case study
3. Design pattern catalog

		Purpose		
		Creational	**Structural**	**Behavioral**
Scope	**Class**	Factory Method (107)	Adapter (class) (139)	Interpreter (243) Template Method (325)
	Object	Abstract Factory (87) Builder (97) Prototype (117) Singleton (127)	Adapter (object) (139) Bridge (151) Composite (163) Decorator (175) Facade (185) Flyweight (195) Proxy (207)	Chain of Responsibility (223) Command (233) Iterator (257) Mediator (273) Memento (283) Observer (293) State (305) Strategy (315) Visitor (331)

Étude de cas [Gamma et al]

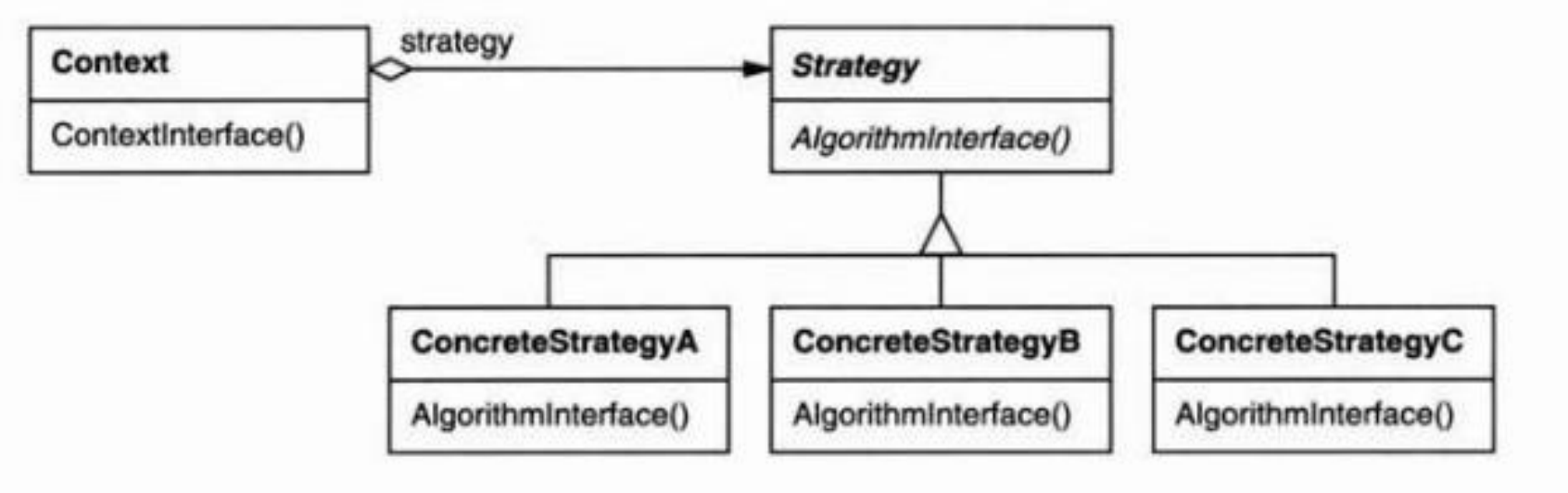
- Traitement de texte
- Document peut contenir, texte, images, figures, tableaux, …
- On peut interagir avec les éléments individuels ou en tant que groupes (ex: colonne)
- Structure interne doit permettre:
  - Calculer formatage
  - Faire l'affichage
  - Retrouver sur quoi on a cliqué lors d'un clic
  - …



Défi 1: Support de plusieurs algorithmes de formatage

- Être en mesure de séparer le texte en plusieurs lignes; décider combien de pixels afficher entre chaque mot
- Différents algorithmes possibles
  - Type de mise en page
  - Qualité vs temps exécution
- L'utilisateur peut décider quel algorithme utiliser
- On veut pouvoir en utiliser des nouveaux
- **Quel patron nous faut-il?**

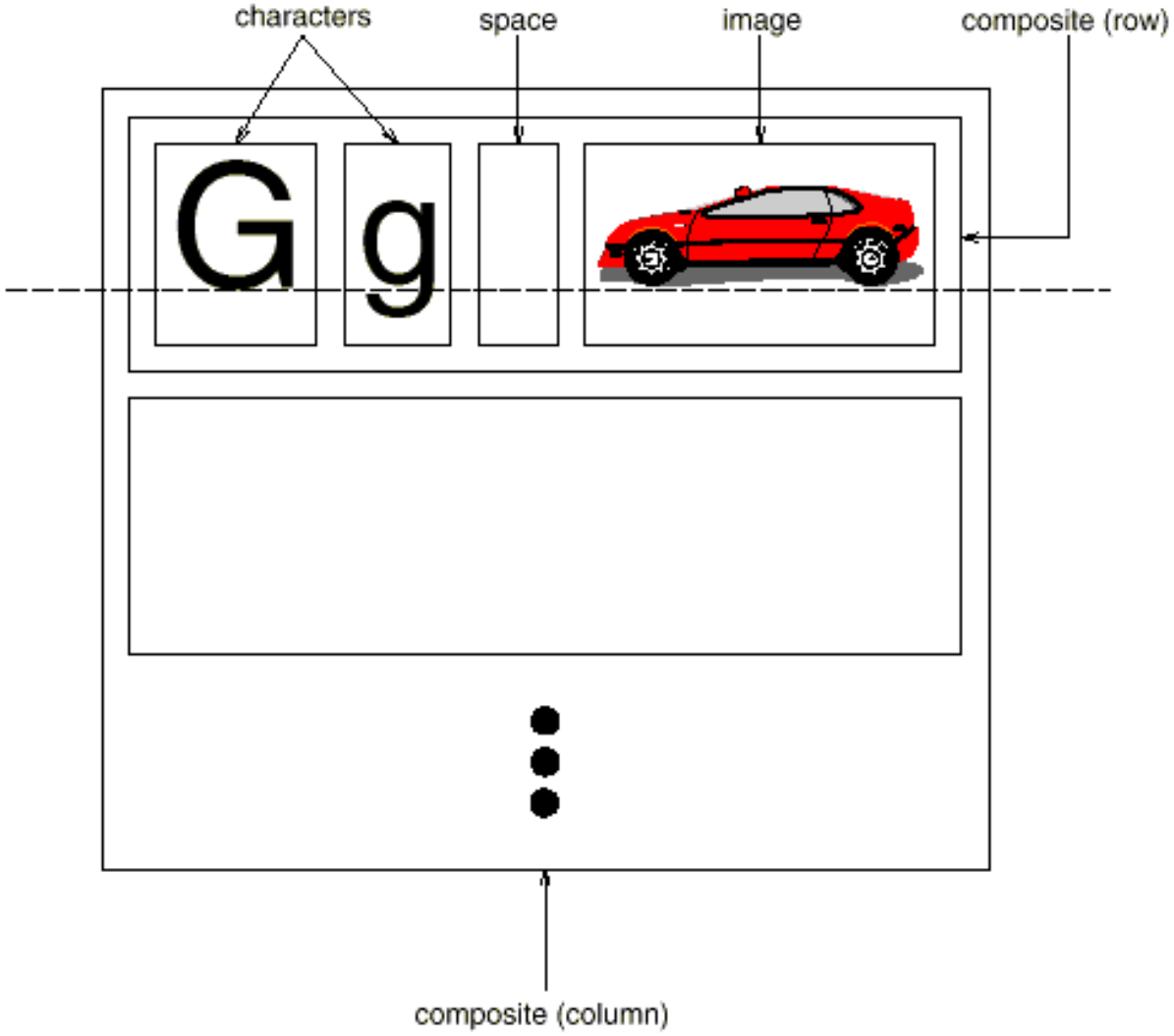
Solution: Strategy



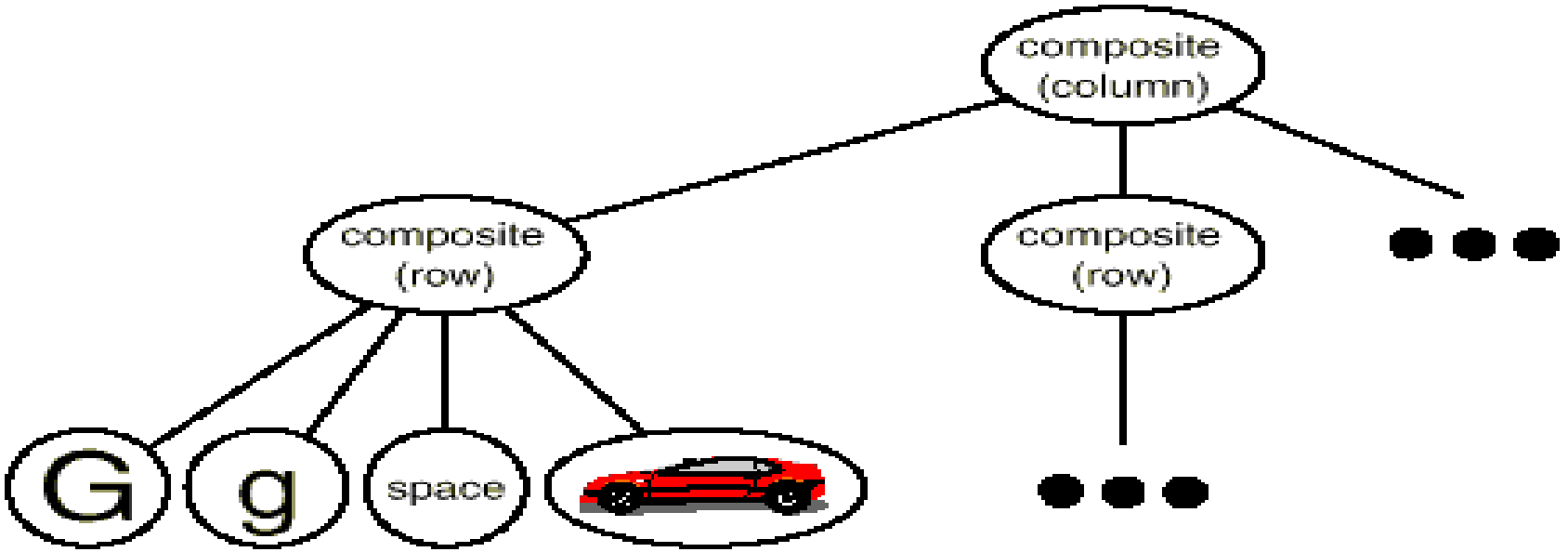
Défi 2: Le document est constitué de plusieurs éléments disparates…

- Éléments graphiques élémentaires (caractères, images, etc.)
- Éléments graphiques plus complexes (ex: tableaux, comportant eux-mêmes des éléments élémentaires)
- On veut autant pouvoir formater une page que l'intérieur d'un tableau en utilisant les mêmes algorithmes
- On veut un système qui fait qu'on pourra ajouter de nouveaux éléments (ex: équations) au fil du temps
- **Quel patron nous faut-il?**

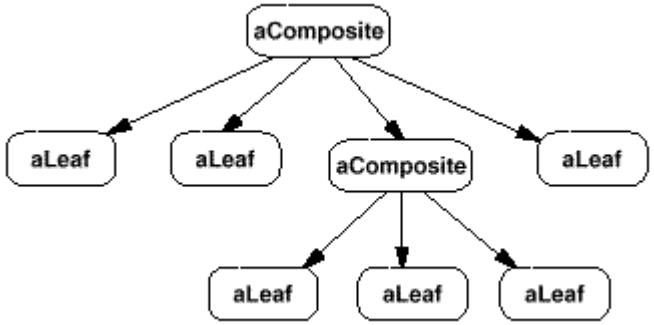
Défi 2: Le document est constitué de plusieurs éléments disparates…

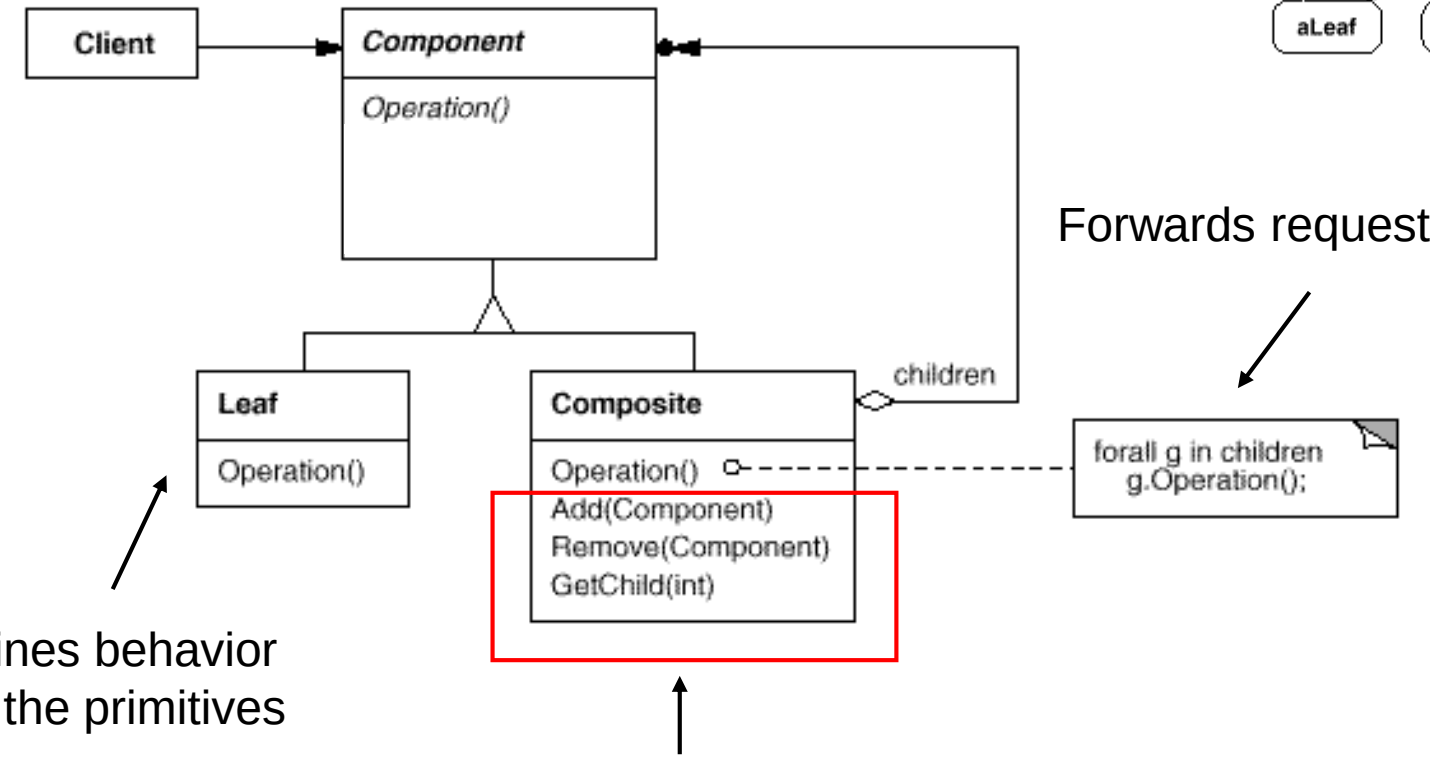


Solution: Composite



Solution: Composite





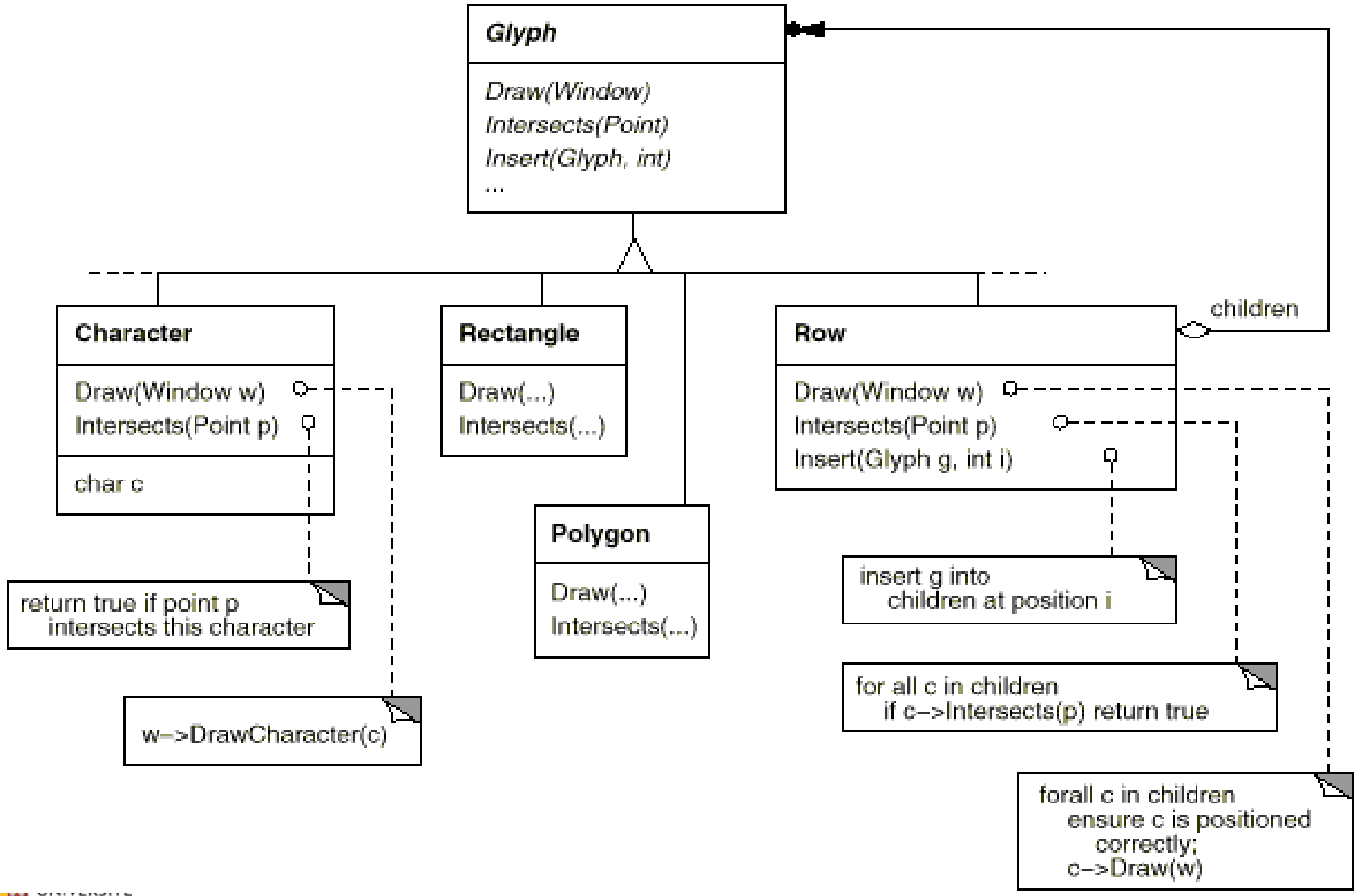
Forwards requests to children

Defines behavior
For the primitives

-Behavior for composites
-Stores Child components

[Lunch & Kidwell, 2008]

Solution: Composite



Défi 3: Différentes versions pour différents systèmes d'exploitation

- La façon d'interagir avec chaque système d'exploitation n'est pas la même
- Il peut exister des version différentes de certaines classes d'un système à l'autre
- **Quel patron nous faut-il?**

Solution: Factory (ou AbstractFactory)

- **Problème**: on veut pouvoir créer des objets sans avoir à spécifier leur classe concrète (le code « appelant » ne connaît pas le nom de la classe à instancier)
- **Solution**: 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();
```

Conclusion sur les patrons

- Facile de faire un mauvais design orienté-objet
- Difficile d'en faire un bon (maintenable, « évoluable », extensible, …)
- Les patrons décrivent des solutions éprouvées que l'on peut exploiter dans nos propres design
- Thèmes récurrent dans les patrons:
  - Faire évoluer l'application en ajoutant des classes, plutôt qu'en modifiant des classes
  - Favoriser l'agrégation plutôt que l'héritage comme mécanisme permettant de faciliter l'évolution future
  - Délégation: un objet reçoit une requête mais fait faire le travail par un autre
  - Ne pas avoir à connaître la classe de l'objet qu'on manipule

À faire cette semaine

- Comprendre à fond les patrons étudiés
  - Lecture des chapitres de Larman
    - Version anglaise: 26
    - Version française: 23
  - Compléments sur ces patrons dans le livre de Gamma (si nécessaire)
- Projet