Conception Orientée Objet

Design patterns – Partie 3

Tianxiao LIU
Master IISC 1ère Année
CY Cergy Paris Université
http://depinfo.u-cergy.fr/~tliu/coo.php

Sommaire : les 23 patterns GoF

Creational	Structural	Behavioral
Abstract Factory	Adapter	**Chain of responsibility**
Builder	Bridge	Command
Factory Method	Composite	**Interpreter**
Prototype	Decorator	Iterator
Singleton	**Facade**	Mediator
	Flyweight	Memento
	Proxy	**Observer**
		State
		Strategy
		Template Method
		Visitor

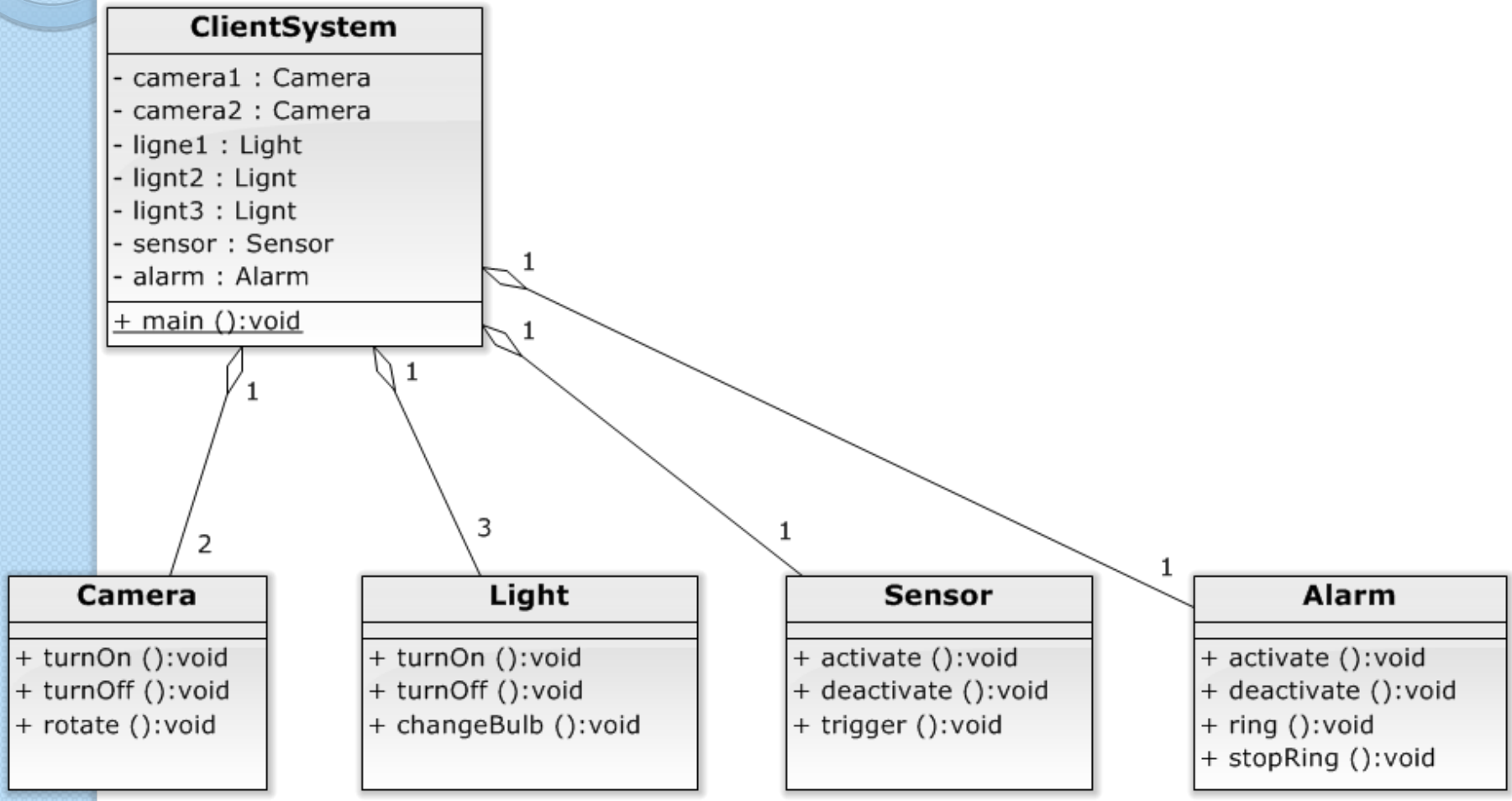
Présentés dans cette séance *Déjà présentés*

Pattern : *Facade*

- Motivation
  - Pour un système complexe, simplifier la communication externe / interne
- Principe
  - Utiliser une ou plusieurs classes *Facade* qui sont responsables de gérer le système
  - A l'extérieur du système, on ne voit que la (les) classe (s) *Facade*

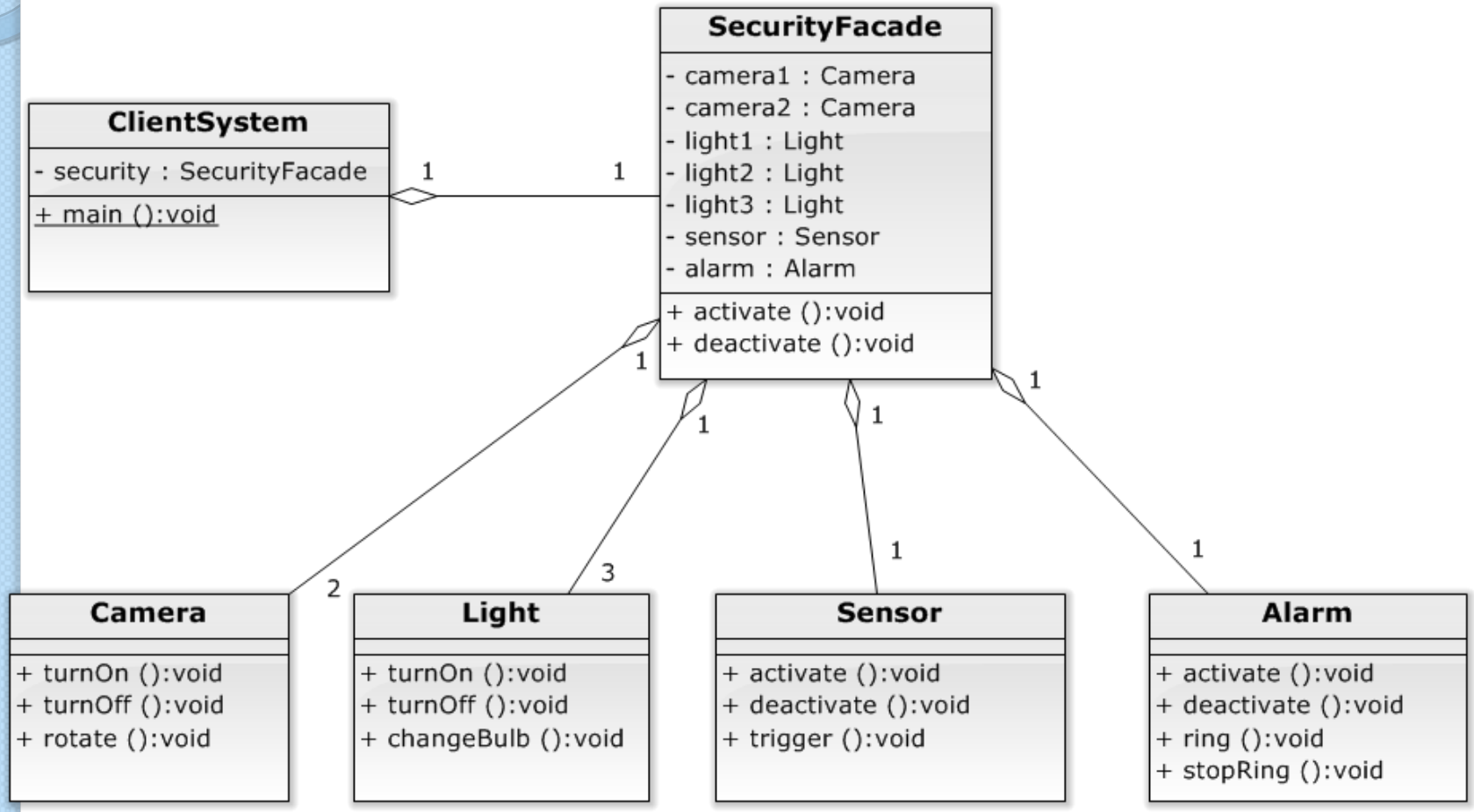
Pattern : *Facade*

- Exemple : Un système de sécurité (**sans utiliser *Facade***)



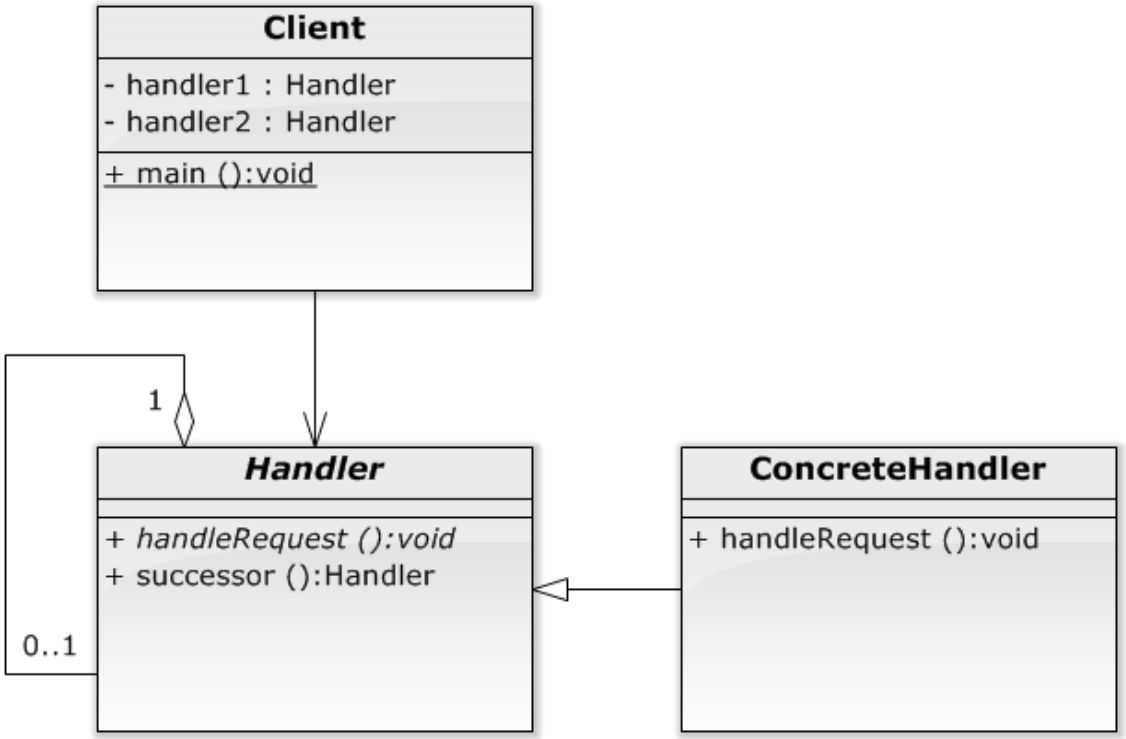
Pattern : *Facade*

- Exemple : Un système de sécurité **(avec pattern *Facade*)**



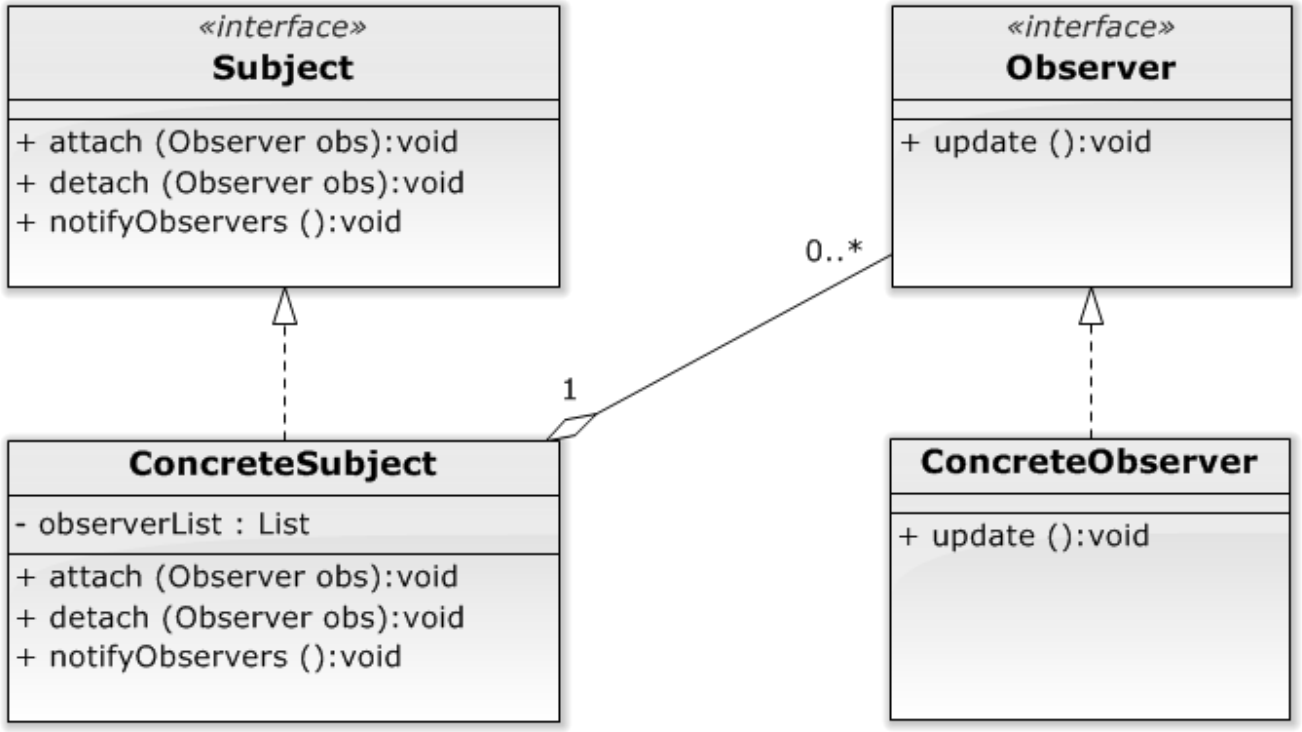
Pattern : *Chain of responsibility*

- Principe
  - On ne sait pas quel objet sur la chaîne va traiter la requête → traitement dynamique



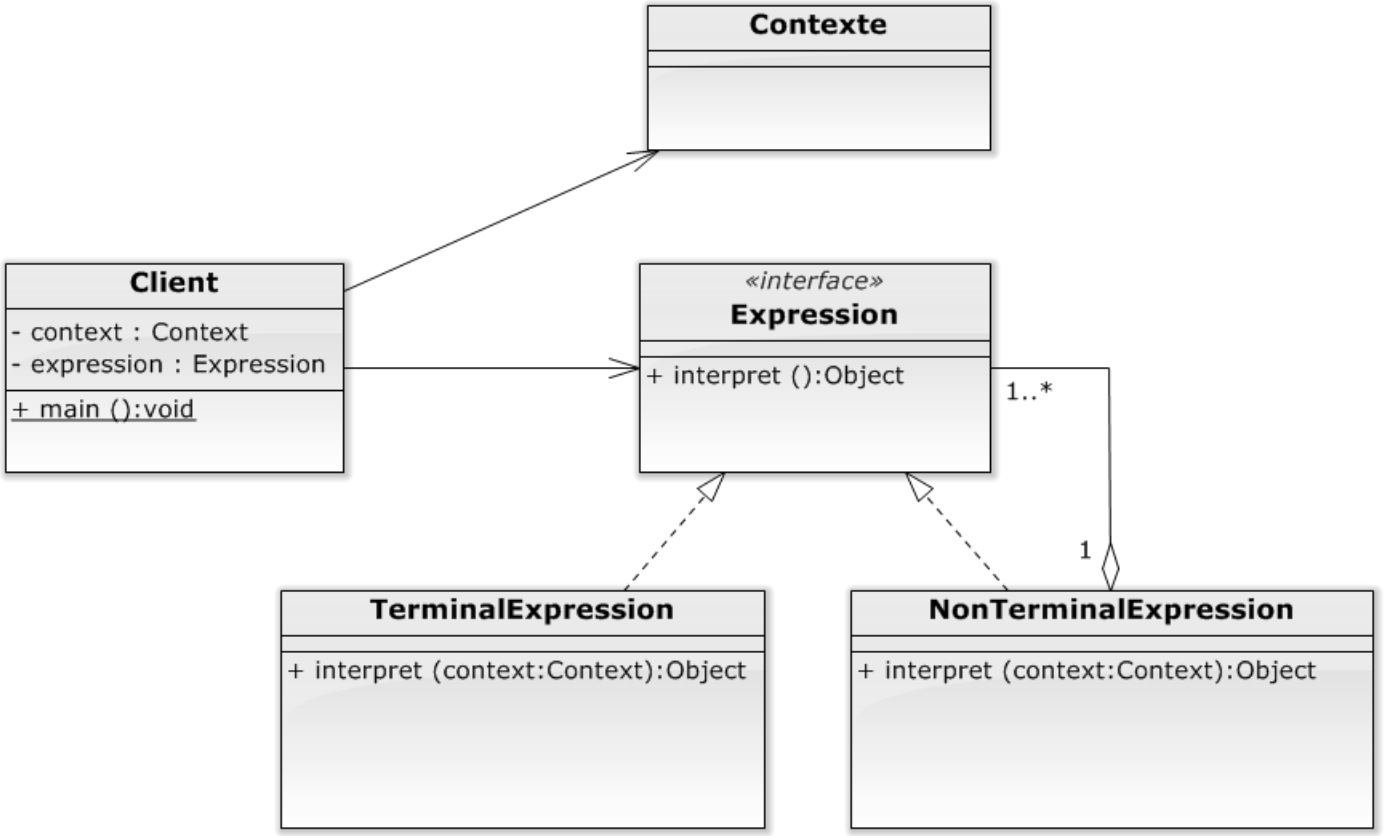
Pattern : *Observer*

- Principe
  - Propagation des mises à jour d'un objet (*observable*) à tous les objets qui l'*observent*



Pattern : *Interpreter*

- Principe
  - Définir un langage et sa syntaxe
  - Calculer (interpréter) les expressions

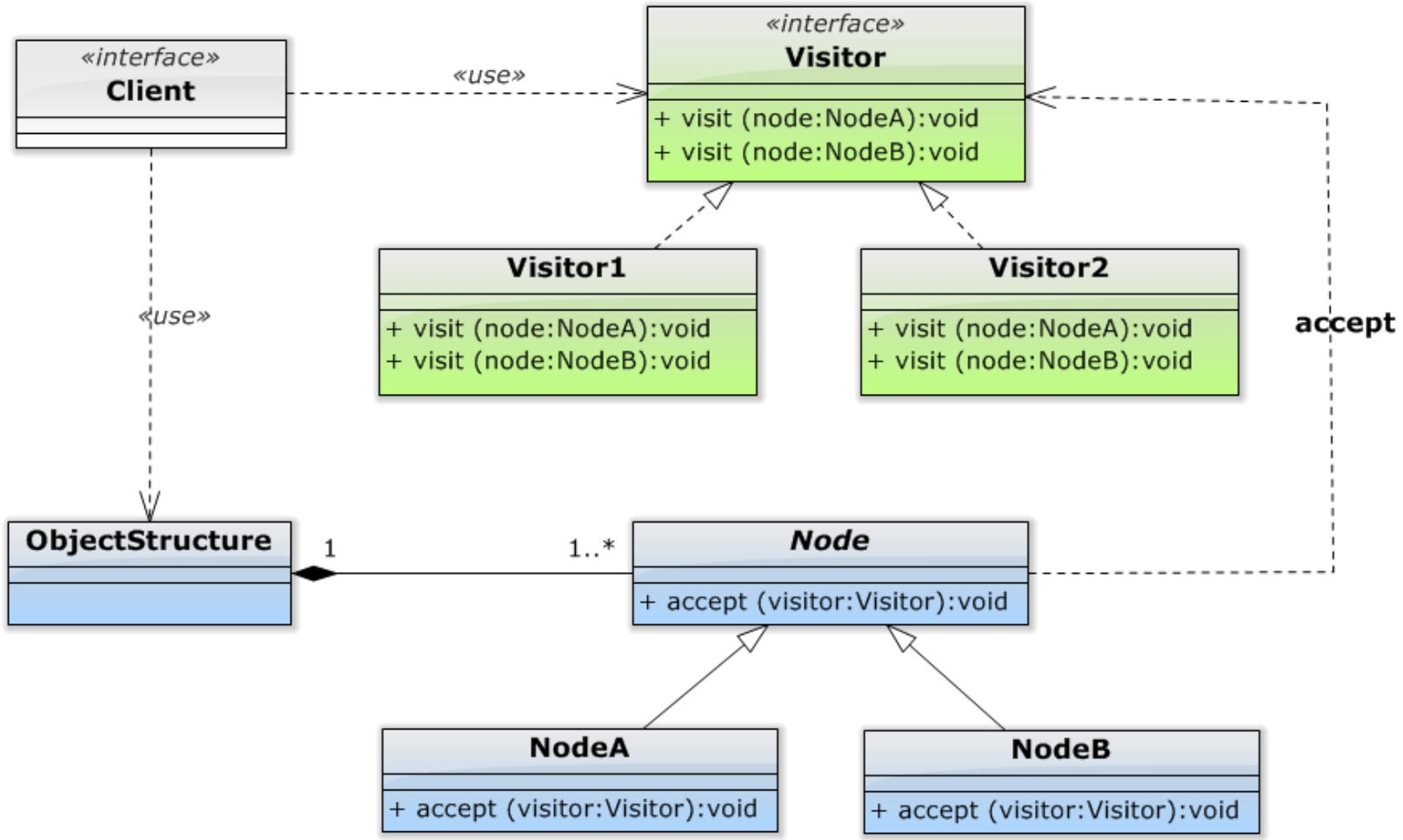


Pattern : *Visitor*

- Principe
  - Découpler les structures de données et les opérations effectuées sur ces structures
- Notions de base
  - *Visitor* abstrait
    - Déclarer une méthode pour chaque type de nœud à visiter → ajout d'un nouveau type de nœud = ajout d'une méthode
  - *Visitor* concret
    - Une implémentation du *visitor* abstrait
  - *Node* abstrait
    - Déclaration de l'acceptation des visites par le *visitor*
  - *Node* concret
    - Un type de nœud concret
  - *ObjectStructure*
    - Un point d'entrée, le créateur de tous les nœuds de la structure

Pattern : *Visitor*

- Diagramme de classes illustrant le modèle



Conseils pour utiliser les patterns

- Ne pas abuser
  - Les design patterns aident à la réutilisabilité
  - Utiliser des patterns = Je suis fort ! ☺ → abus ! ☹
- Principes à respecter
  - Analyser le problème à résoudre
  - Comprendre les patterns
  - Réflexion sur les avantages et les inconvénients

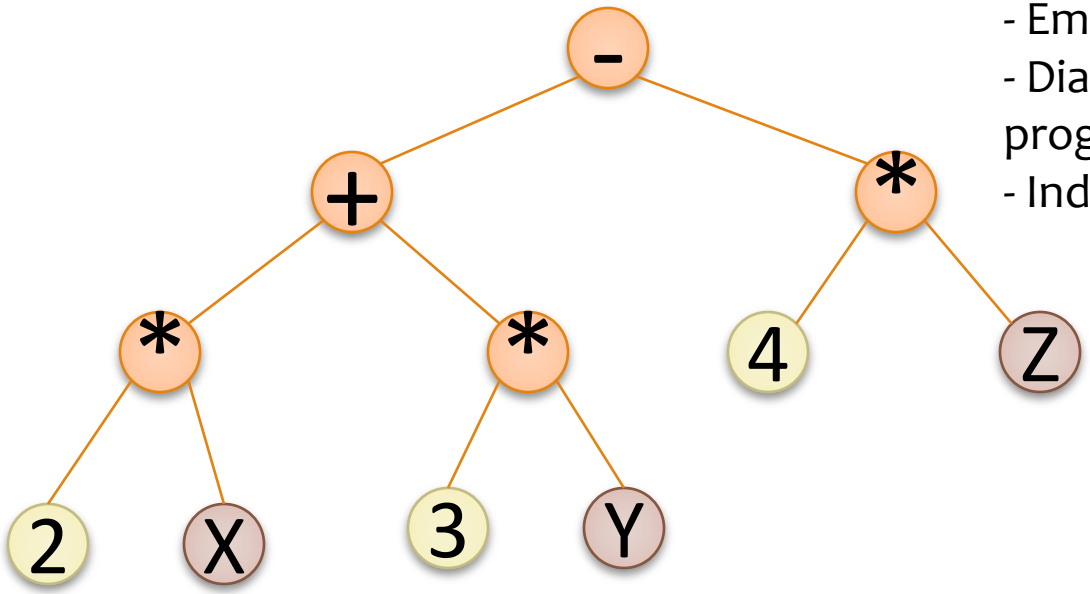
Patterns : application des principes d'objet

Patterns	OCP	LSP	DIP	ISP	CARP	LoD
Singleton						
Factory method	x		x			
Abstract Factory	x					
Prototype						
Builder	x			x		x
Adapter	x				x	
Bridge	x				x	
Composite	x	x			x	
Decorator	x				x	
Façade	x			x		x
Flyweight						
Proxy		x			x	x
Chain of responsibility	x					
Interpreter	x	x			x	
Iterator	x		x	x		
Observer	x		x			x
Strategy	x	x			x	
Template method	x		x			
Visitor	x		x			

OCP : Open Close Principle
LSP : Liskov Substitution Principle
DIP : Dependency Inversion Principle
ISP : Interface Segregation Principle
CARP : Composition / Aggregation Reuse Principle
LoD : Law of Demeter

Programme Tree – nouvelle version

- Calcul de formule arithmétique
  - Exemple : `2*X + 3*Y – 4*Z`
  - Représentation : arbre binaire



Devoir à rendre :
- Dimanche **18 octobre 2020 21H**
- Email intitulé « **COO Tree** »
- Diagramme de classe du programme Tree : **image**
- Indiquez les patterns utilisés