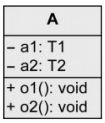
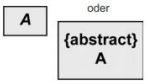
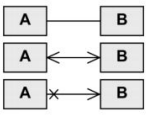
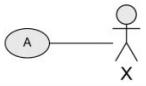
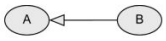
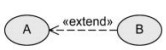
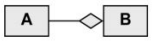
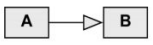
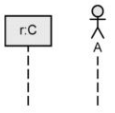
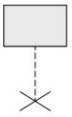
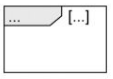
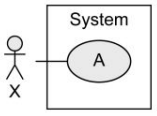
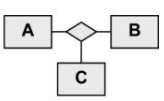
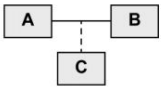
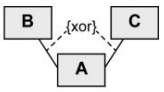
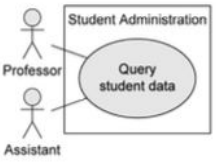
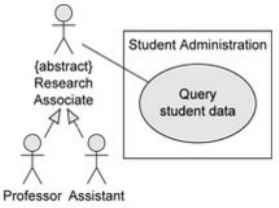
Name	Notation	Description
Class		Description of the structure and behavior of a set of objects
Abstract class		Class that cannot be instantiated
Association		Relationship between classes: navigability unspecified, navigable in both directions, not navigable in one direction
Synchronous message		Sender waits for a response message
Response message		Response to a synchronous message
Asynchronous communication		Sender continues its own work after sending the asynchronous message
Lost message		Message to an unknown receiver
Found message		Message from an unknown sender
Association		Relationship between use cases and actors
Generalization		Inheritance relationship between actors or use cases
Extend relationship		B extends A: optional use of use case B by use case A
Include relationship		A includes B: required use of use case B by use case A
Shared aggregation		Parts-whole relationship (**A** is part of **B**)
Strong aggregation = composition		Existence-dependent parts-whole relationship (**A** is part of **B**)
Generalization		Inheritance relationship (**A** inherits from **B**)
Object		Instance of a class
Link		Relationship between objects

Lifeline		Interaction partners involved in the communication
Destruction event		Time at which an interaction partner ceases to exist
Combined fragment		Control constructs
System		Boundaries between the system and the users of the system
Use case		Unit of functionality of the system
Actor		Role of the users of the system
n-ary association		Relationship between n (here 3) classes
Association class		More detailed description of an association
xor relationship		An object of **C** is in a relationship with an object of **A** or with an object of **B** but not with both

- Example:



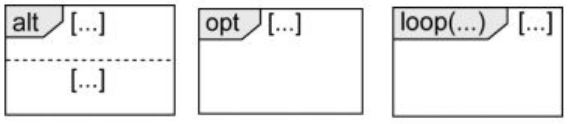
≠



Professor AND **Assistant** needed for executing **Query student data**

Professor OR **Assistant** needed for executing **Query student data**

Combined fragment :



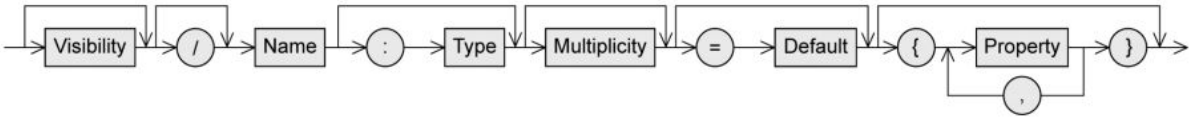
entre crochets : cas d'arrêt (guard). loop(min,max)

Description of Use Cases

Pour standard : (1) … (2)....
Pour Alternative : (1')... (1'')....

- Structured approach
  - Name
  - Short description
  - Precondition: prerequisite for successful execution
  - Postcondition: system state after successful execution
  - Error situations: errors relevant to the problem domain
  - System state on the occurrence of an error
  - Actors that communicate with the use case
  - Trigger: events which initiate/start the use case

Attribute Syntax



+ ... public: everybody
- ... private: only the object itself
... protected: class itself and subclasses
~ ... package: classes that are in the same package

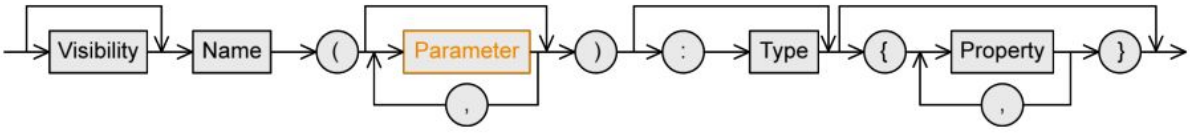
/ : calculé à partir d'autres attributs
Multiplicité : notation : [min..max]
Default : password: String = "pw123"

- Pre-defined properties
  - {readOnly} … value cannot be changed
  - {unique} … no duplicates permitted
  - {non-unique} … duplicates permitted
  - {ordered} … fixed order of the values
  - {unordered} … no fixed order of the values
- Attribute specification
  - Set: {unordered, unique}
  - Multi-set: {unordered, non-unique}
  - Ordered set: {ordered, unique}
  - List: {ordered, non-unique}

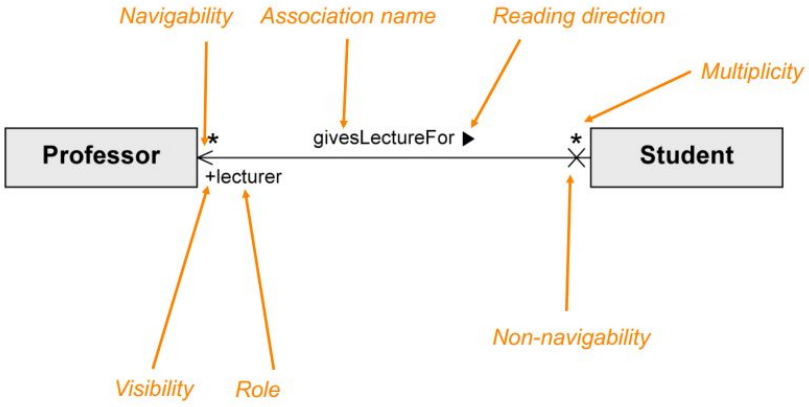
Operations :

- Notation similar to attributes
- Direction of the parameter
  - in … input parameter
    - When the operation is used, a value is expected from this parameter
  - out … output parameter
    - After the execution of the operation, the parameter has adopted a new value
  - inout : combined input/output parameter

Operation Syntax - Parameters

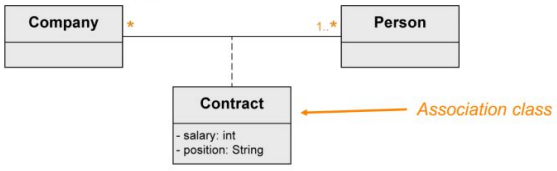


SI C'EST STATIQUE C'EST SOULIGNE



Association Class

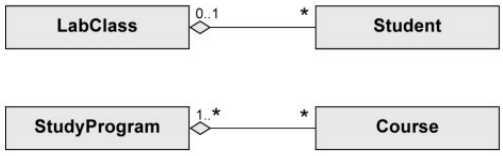
- Necessary when modeling n:m Associations



- Default: no duplicates
- non-unique: duplicates allowed

Shared aggregation :

- **`Student`** is part of **`LabClass`**
- **`Course`** is part of **`StudyProgram`**



Composition :

- Example: **`Beamer`** is part of **`LectureHall`** is part of **`Building`**

If the `Building` is deleted, the `LectureHall` is also deleted

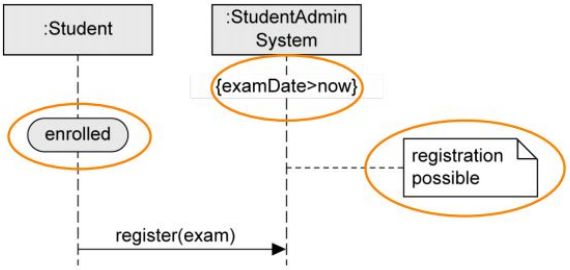
The `Beamer` can exist without the `LectureHall`, but if it is contained in the `LectureHall` while it is deleted, the `Beamer` is also deleted

Time Constraints

- Types
  - Point in time for event occurrence
    - Relative: e.g., **`after(5sec)`**
    - Absolute: e.g., **`at(12.00)`**
  - Time period between two events
    - **`{lower..upper}`**
    - E.g., **`{12.00..13.00}`**

State invariant :

- Three alternative notations:




```
@tag @tag                                          optional
Feature:
description
Background:
   Given
   And
@tag @tag
Scenario:
   Given
      And
      But
   When
      And
      But
   Then
      And
      But
```

Background step(s)
Additional contextual information. Runs before each scenario.

Given block
Get system into known starting state

When block
Key actions the user performs

Then block
Observable business-value system output

Given the following users exist:

name	age
Sarah	42
Gentry	28
Amrit	34

A DataTable will be passed to the underlying automation code containing 3 rows of data & 2 columns: name & age

Scenario Outline: addition
Given there is currently <start>
When I add <num>
Then I should end up with <result>

Examples:

start	num	result
0	5	5
20	20	40

The Scenario Outline will execute once for every row in the Examples table

Patterns :

– Extensibilité
– Flexibilité
– Maintenabilité
– Réutilisabilité + renforcer la cohésion et diminuer le couplage

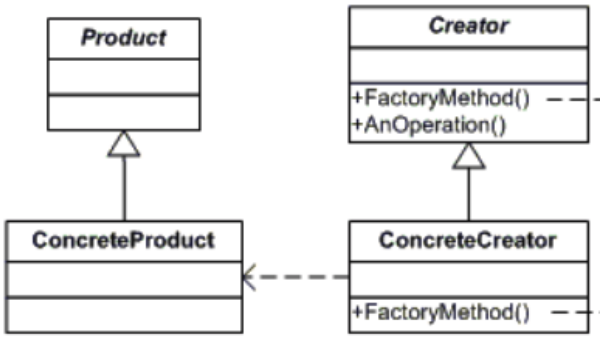
Patrons

Renforcer cohésion, diminuer couplage.

Objectif				
		Création	Structure	Comportement
Portée	Classe	*Factory Method*	*Adapter*	*Interpreter* *Template Method*
	Objet	*Abstract Factory* *Builder* *Prototype* *Singleton*	*Adapter* *Bridge* *Composite* *Decorator* *Facade* *Flyweight* *Proxy*	*Chain of Responsibility* *Command* *Iterator* *Mediator* *Memento* *Observer* *State* *Strategy* *Visitor*

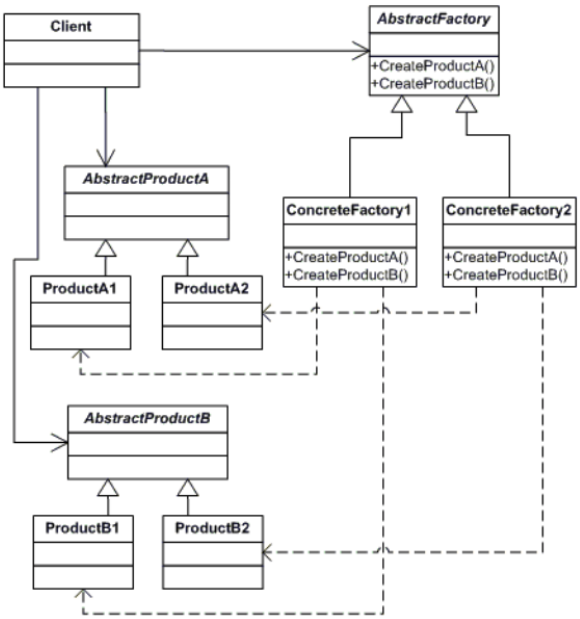
Factory Method

- Définit une interface pour la création d'un objet, mais en laissant à des sous-classes le choix des classes à instancier.
- Permet à une classe de déléguer l'instanciation à des sous- classes.
- Souvent des singletons



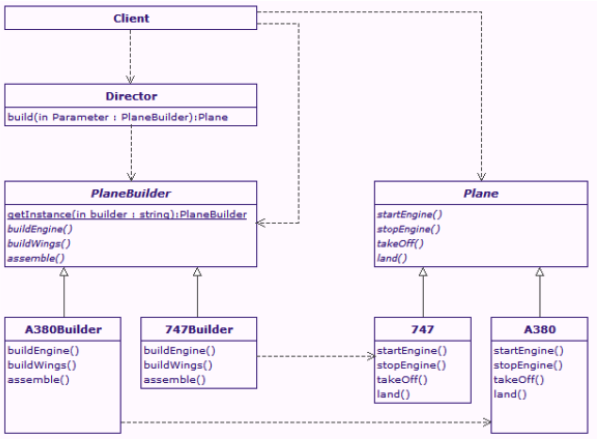
Abstract Factory

– Fournir une interface pour créer des familles d'objets dépendants ou associés sans connaître leur classe réelle.



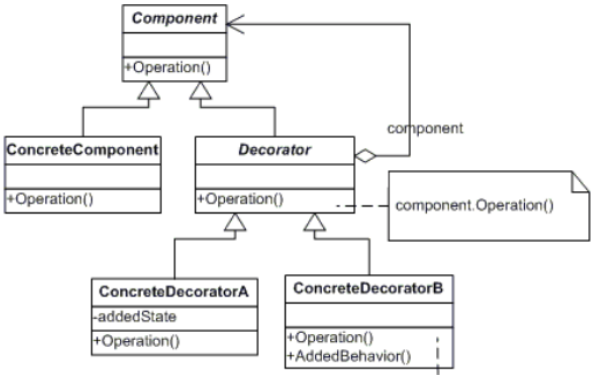
Builder

– Séparer la construction d'un objet complexe de sa représentation.
– De sorte qu'un même processus de construction puisse créer différentes représentations.



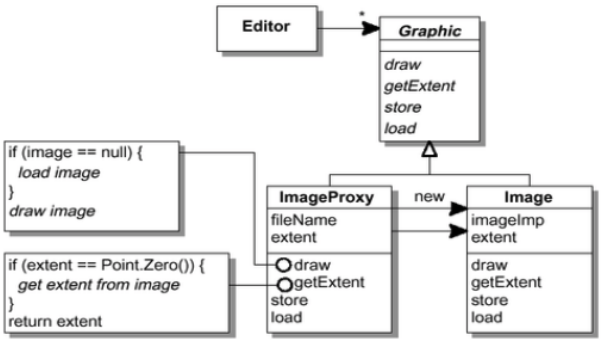
Decorator

– Attacher dynamiquement des capacités additionnelles à un objet.
– Fournir ainsi une alternative flexible à l'héritage pour étendre les fonctionnalités.



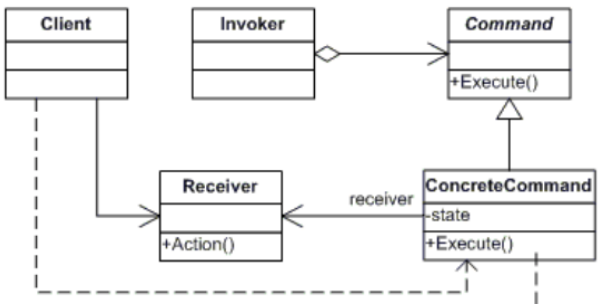
Proxy

– Fournir un substitut afin d'accéder à un autre objet souvent inaccessible.
– Séparer l'interface de l'implémentation



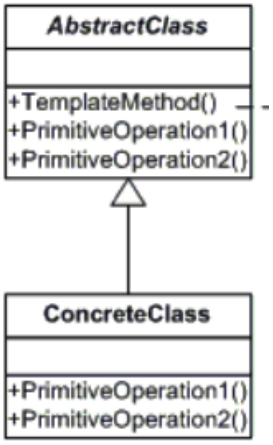
Command

– Encapsuler une requête comme un objet
– Permettre de défaire des traitements (undo)



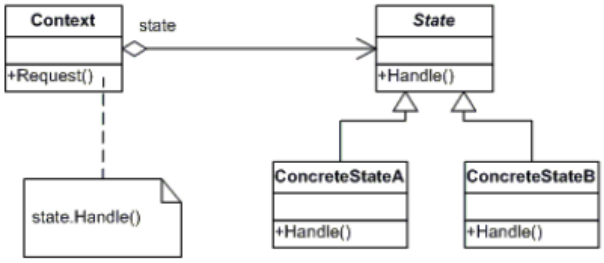
Template Method

– Définir le squelette d'un algorithme dans une opération, et laisser les sous-classes définir certaines étapes.
– Permet de redéfinir des parties de l'algorithme sans avoir à modifier celui-ci.



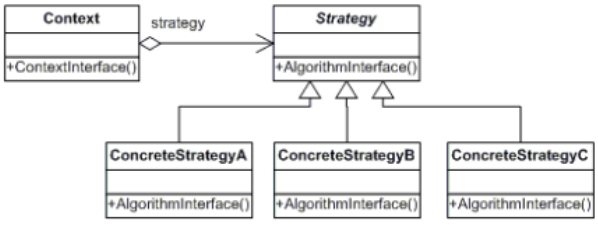
State

- Modifier le comportement d'un objet quand son état interne change.
- Obtenir des traitements en fonction de l'état courant.
- Tout est mis en place pour donner l'impression que l'objet lui-même a été modifié



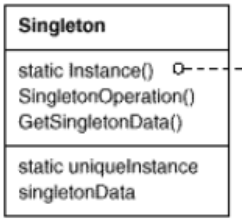
Strategy

– Définir une hiérarchie de classes pour une famille d'algorithmes, encapsuler chacun d'eux, et les rendre interchangeables.
– Les algorithmes varient indépendamment des clients qui les utilisent



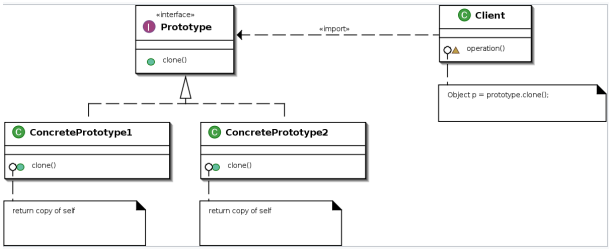
Singleton

– S'assurer qu'une classe a une seule instance, et fournir un point d'accès global à celle-ci.



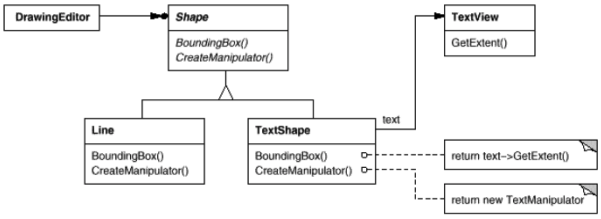
Prototype

– Indiquer le type des objets à créer en utilisant une instance (le prototype). les nouveaux objets sont des copies de ce prototype (clonage).



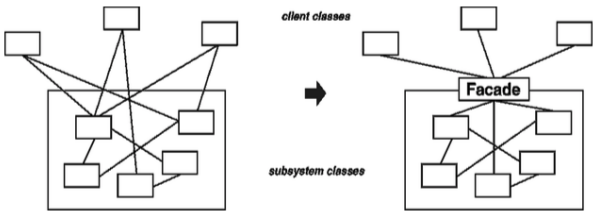
Adapter

– Convertir l'interface d'une classe en une autre interface qui est attendue par un client.
– Permet de faire collaborer des classes qui n'auraient pas pu le faire à cause de l'incompatibilité de leurs interfaces.



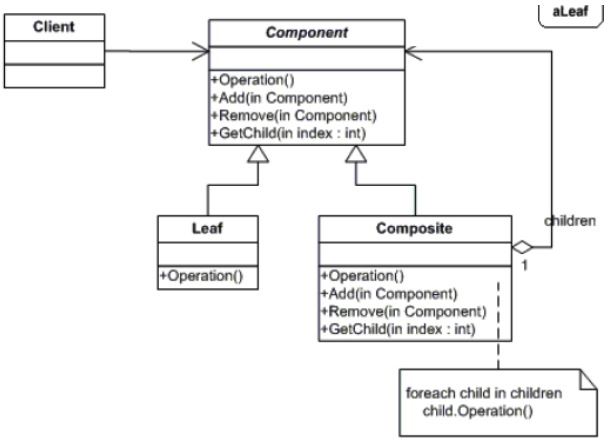
Façade

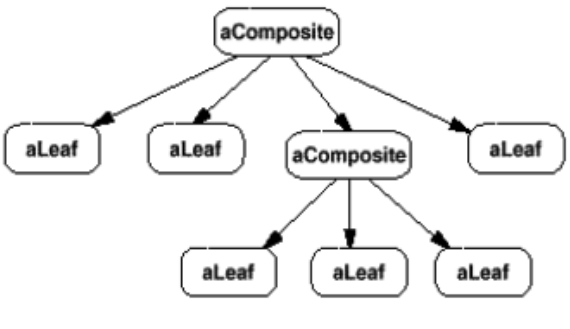
– Fournir une interface unique, simplifiée ou unifiée, pour accéder à un ensemble d'interfaces d'un sous-système complexe.



Composite

– Composer des objets dans des structures d'arbre pour représenter des hiérarchies composants/composés.
– Composite permet au client de manipuler uniformément les objets simples et leurs compositions





Bridge

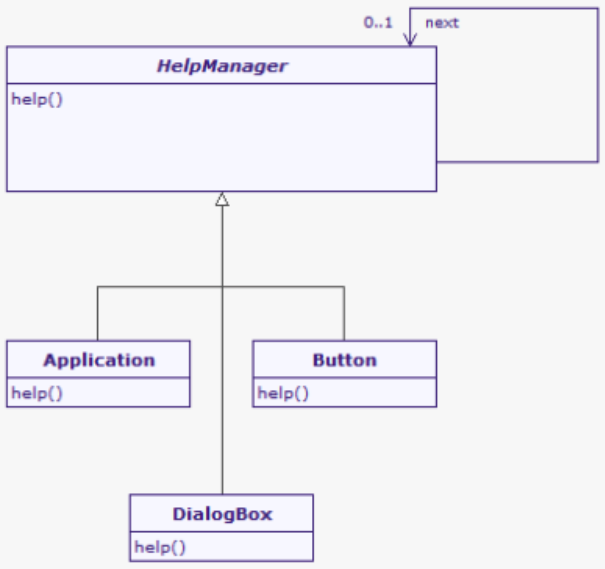
– Découple l'abstraction de l'implémentation afin de permettre aux deux de varier indépendamment.
– Partager une implémentation entre de multiples objets.
– En Java, programmation par deux interfaces

Flyweight

– Utiliser une technique de partage qui permet la mise en œuvre efficace d'un grand nombre d'objets de fine granularité.
– Distinction entre état intrinsèque et état extrinsèque

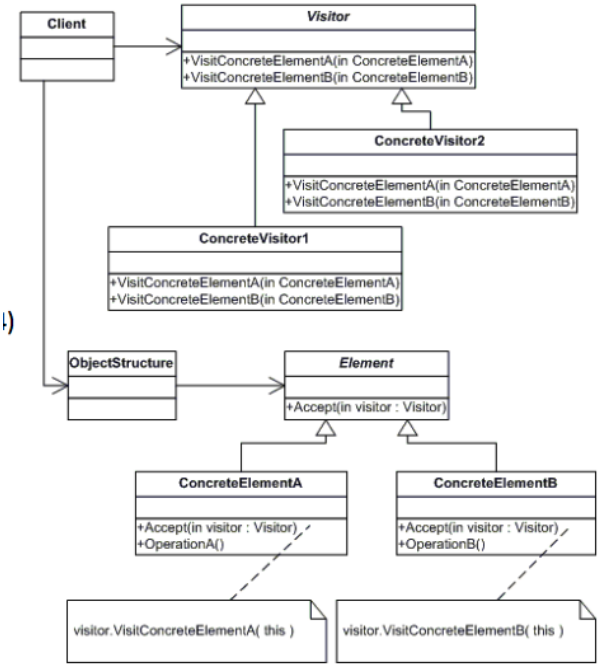
Chain of Responsibility

– Permettre à un objet d'envoyer une instruction (requête) sans savoir quel objet va effectuer le traitement.
– Faire suivre une demande le long de la chaîne jusqu'à ce qu'elle soit traitée par un récepteur.



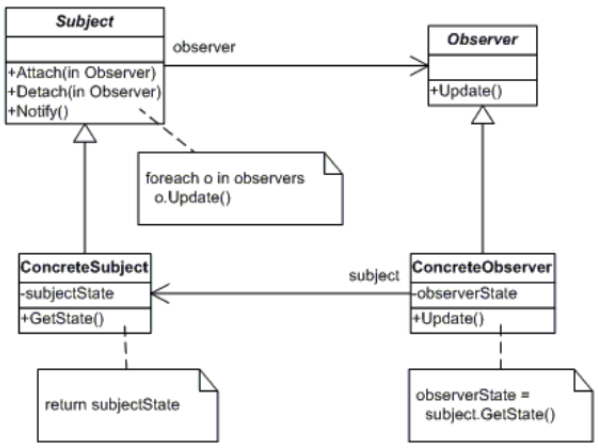
Visitor

– Représenter UNE opération à effectuer sur les éléments d'une structure.
– Permet de définir une nouvelle opération sans changer les classes des éléments sur lesquels on opère.



Observer

Définir une dépendance 1-N de telle façon que si l'objet change d'état tous ses dépendants sont prévenus et mis à jour automatiquement.



Other

Interprète

– Pour un langage donné, définir une représentation pour sa grammaire, fournir un interprète capable de manipuler ses phrases grâce à la représentation établie

Iterator

Mediator

– Encapsule les modalités d'interaction d'un certain ensemble d'objets
– Couplage faible en dispensant les objets de se faire explicitement référence

Mémento

– Externalise, enregistre (puis restaure) l'état d'un objet