ULB UNIVERSITÉ LIBRE DE BRUXELLES

INFO-H-301
Programmation orientée objet

TP 7 :

Introduction à UML

Bonnes pratiques et *Design Patterns* (1)

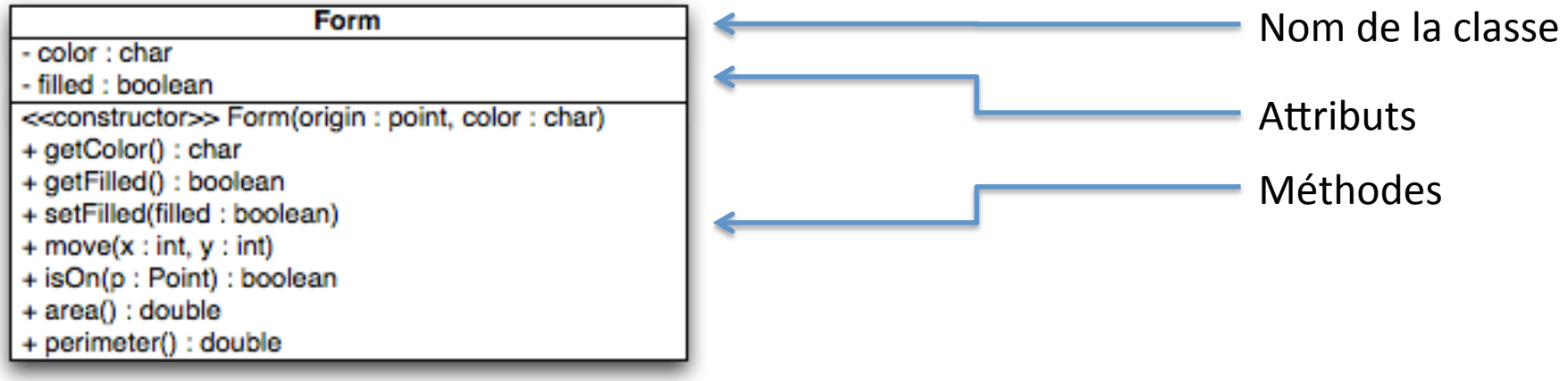
ULB UNIVERSITÉ LIBRE DE BRUXELLES

CODE

Partie 1 : Introduction à UML

Diagrammes de classes et de séquences

Classes, méthodes et attributs



Encapsulation :

```
+ : public
- : private
# : protected
~ : package-private
```

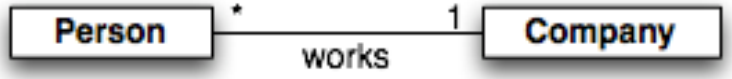
Les types sont précisés à l'aide d'un " : "

Le constructeur est précédé par `<<constructor>>`

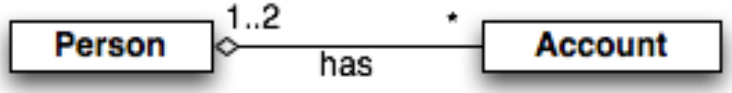
Les membres statiques sont soulignés

Association, composition, agrégation

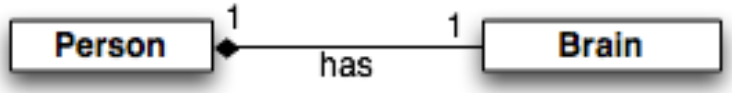
Association : Une personne travaille pour une et une seule compagnie



Agrégation : Une personne possède entre 0 et n comptes



Composition : Une personne a un et un seul cerveau



Cardinalités :

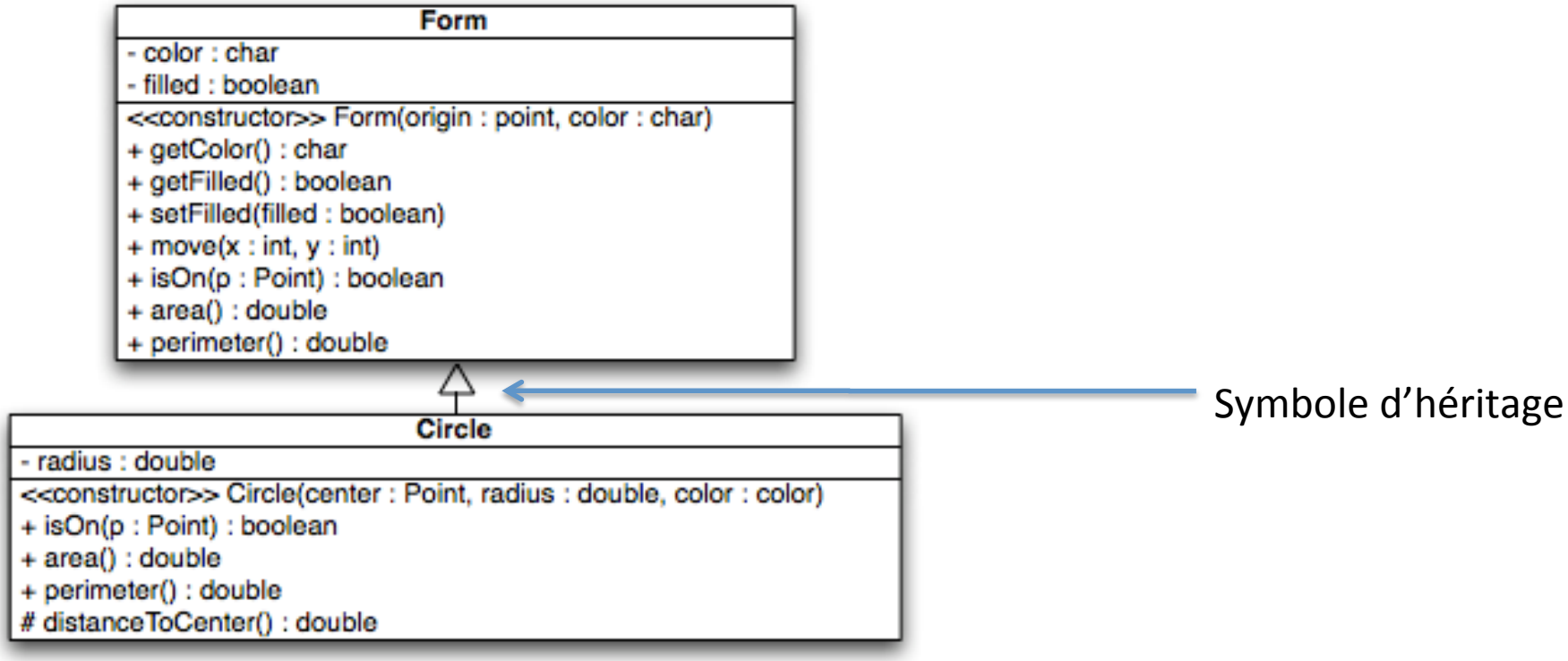
*, *n*, *m*..*, où *n>0* et *m>=0*

Par défaut, la cardinalité est *un*.

Dans une composition, la cardinalité du côté de l'agrégat ne peut être que 1 ou 0..1

Le nom de l'association est facultatif

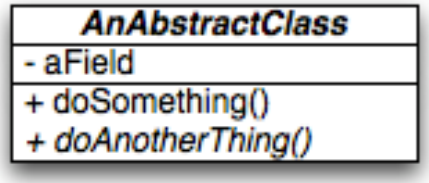
Héritage



Les méthodes redéfinies sont réécrites dans la classe fille ainsi que les méthodes et attributs supplémentaires.

Classes abstraites et interfaces

Classes abstraites :
Nom des classes et méthodes abstraites en italique



Interfaces :
Stéréotype <<interface>> avant le nom de l'interface.

Symbole d'implémentation en pointillés.

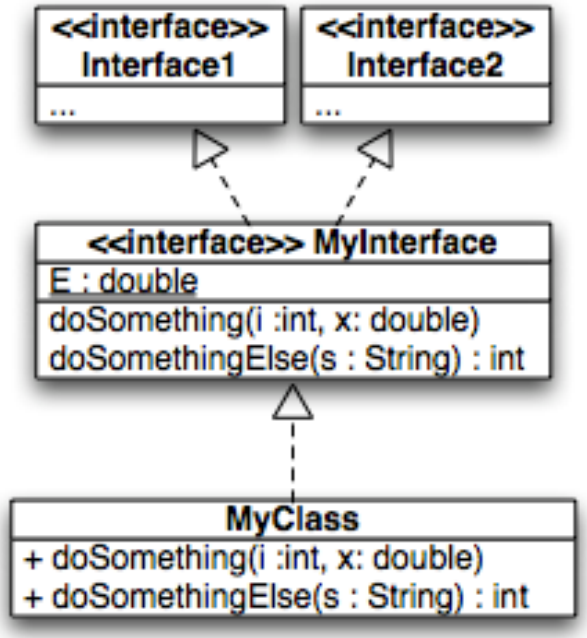


Diagramme de classes UML

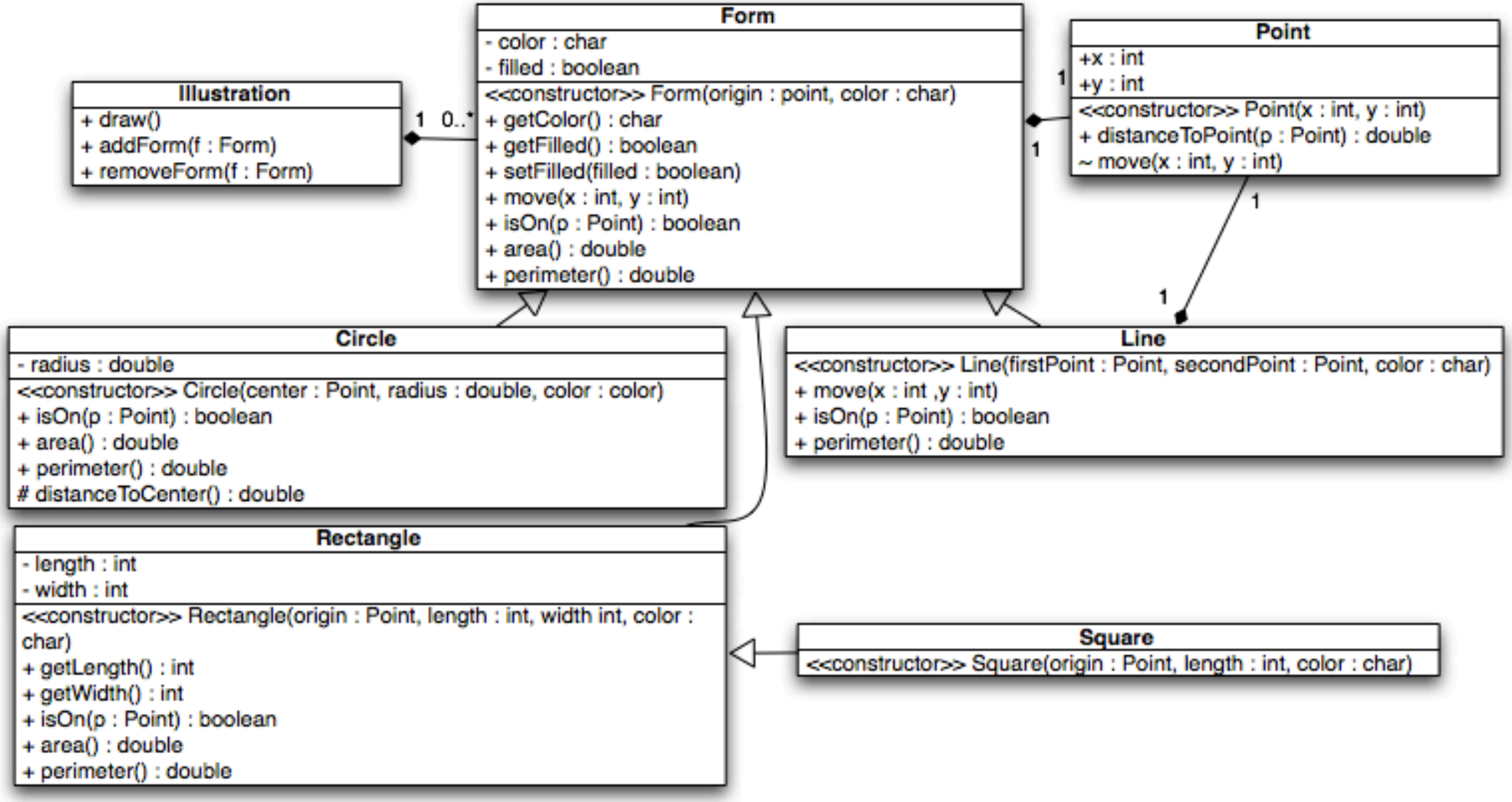
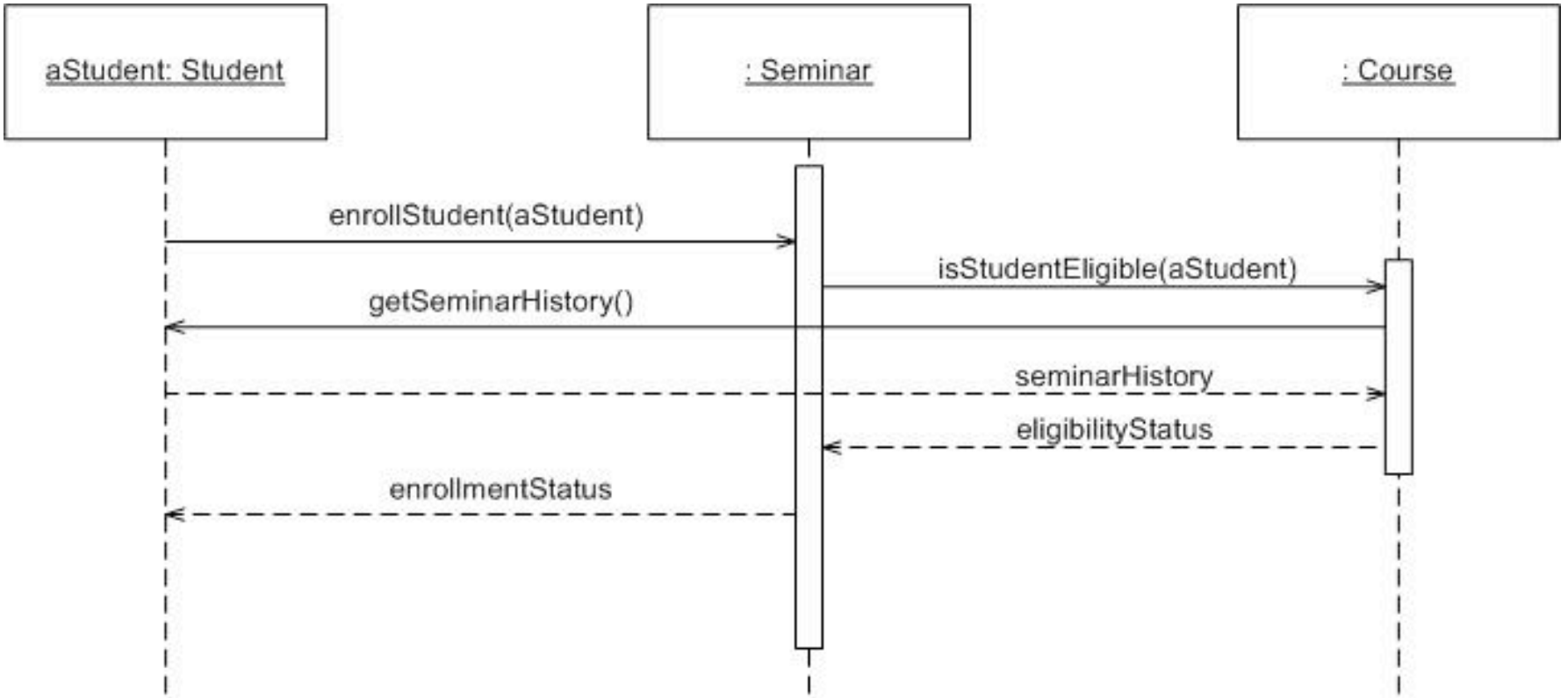


Diagramme de séquences

Permet de représenter un scénario d'envoi de messages entre objets (plus de détails dans le cours).



Partie 2 : Règles de bonnes pratiques OO

Bonnes pratiques de conception OO

"*Controlling complexity is the essence of computer programming.*" (Brian Kernighan)

Quelques principes :

- Expert d'information (celui qui sait le fait)
- Couplage faible
- Forte cohésion
- Loi de Demeter (*Don't talk to strangers*)

Ces principes sont tirés des principes GRASP [1].

Couplage et cohésion

Couplage faible

Le **couplage** mesure la **dépendance** entre des classes.

Dépendance : héritage, attributs, arguments, …

Faible : réduit l'impact des changements dans une classe.

Forte cohésion

La **cohésion** mesure la **compréhensibilité** des classes.

Pour être compréhensible, une classe doit avoir des **responsabilités** cohérentes.

Fort : faciliter la maintenance et l'utilisation d'une classe.

Ces deux règles sont liées.

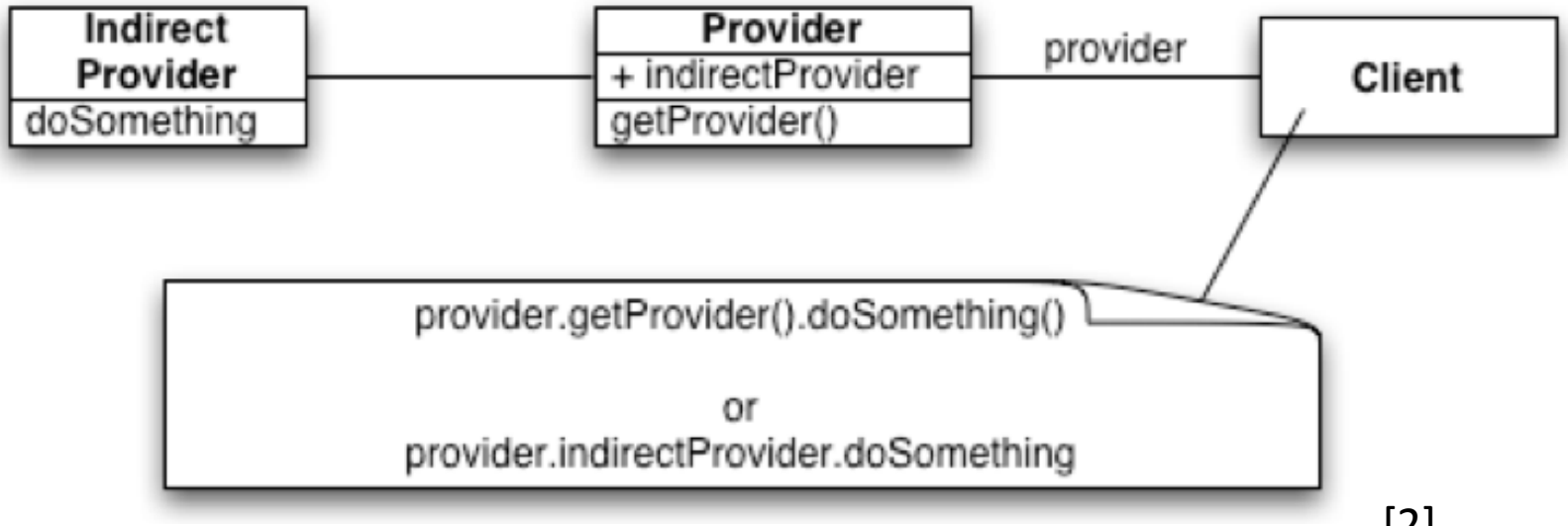
Loi de Demeter

Cas particulier du couplage faible.

La loi dit qu'on ne peut envoyer des messages qu'à :

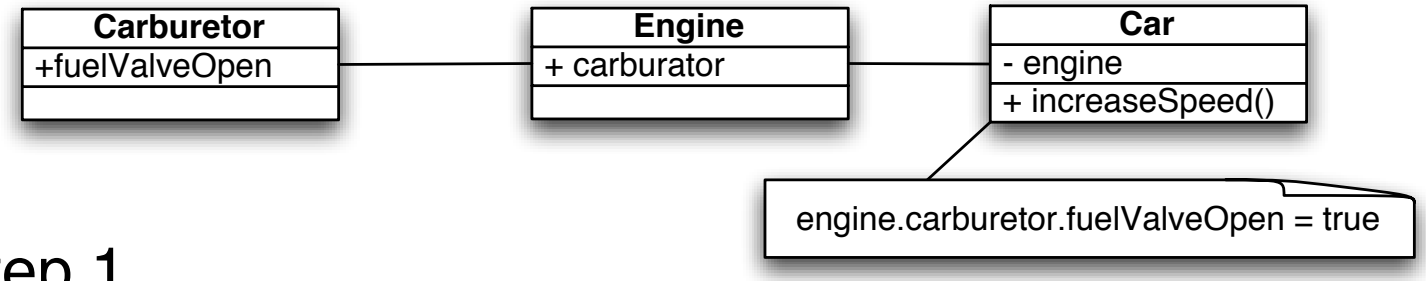
- un argument passé
- un objet que l'on a créé
- self et super

Problème :

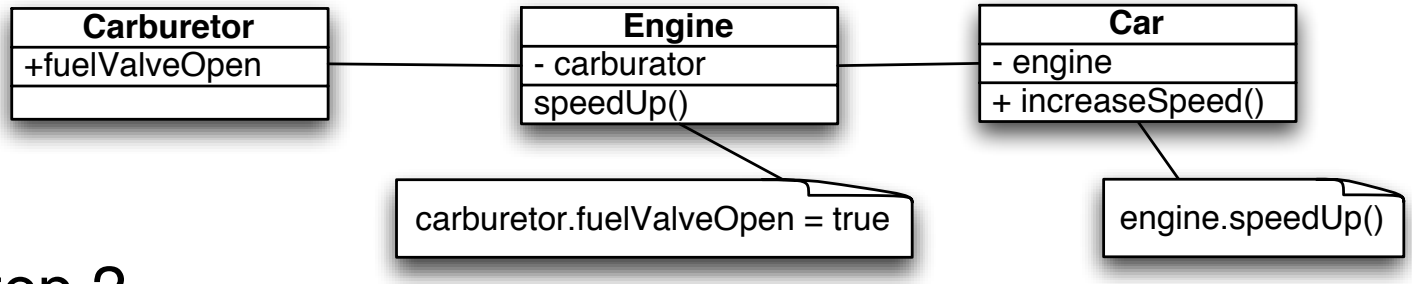


[2]

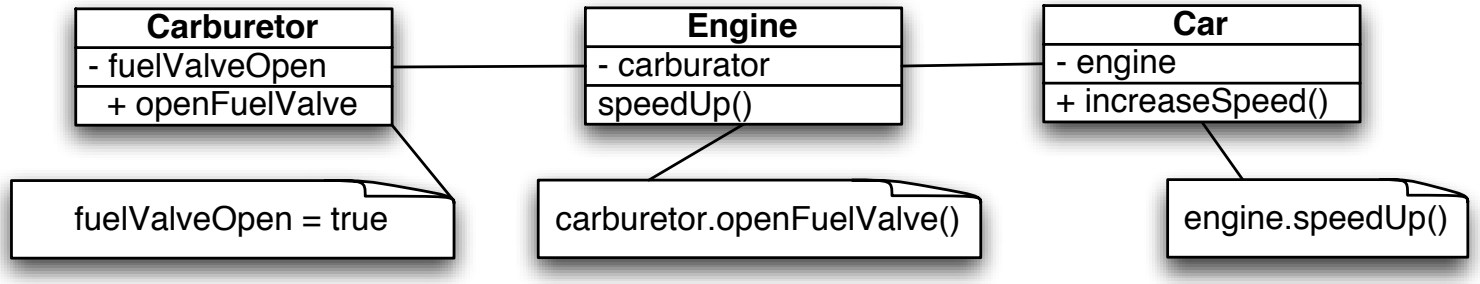
Loi de Demeter : solution



Step 1



Step 2



[2]

ULB UNIVERSITÉ LIBRE DE BRUXELLES

CODE

Partie 3 : *Design Patterns* (I)

Patrons de conception

Concept de génie logiciel visant à résoudre des problèmes récurrents d'architecture et de conception logiciel.

Formalisation de bonnes pratiques

Capitalisation de l'expérience appliquée à la conception

Formalisés en 1995 par le « Gang of four » [2]

Aussi appellés « Design patterns » ou « Motifs de conception ».

Patrons de conception GoF

Trois grandes familles :

Patrons de construction

instancier et configurer des classes et des objets

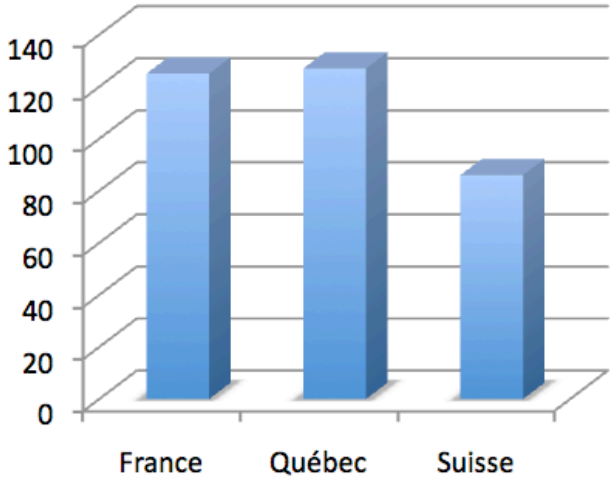
Patrons structuraux

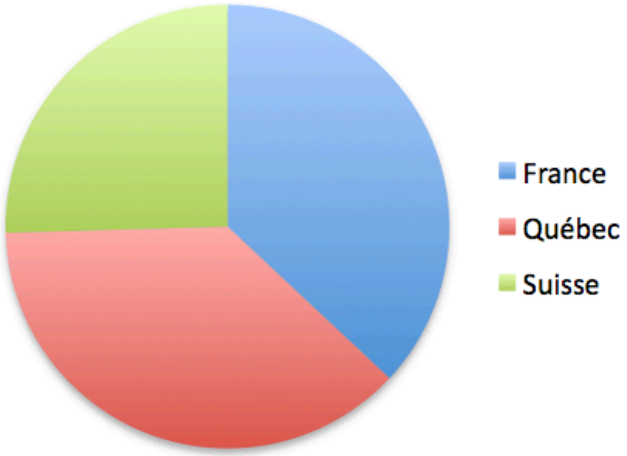
organiser les classes d'un programme en séparant l'***interface*** de l'implémentation

Patrons comportementaux

organiser les objets pour que ceux-ci ***collaborent***

Patron de conception **Observer**





France	125
Québec	127
Suisse	86

Patron de conception Observer

Problème

Plusieurs objets A, les observateurs, sont notifiés lorsqu'un objet B, le sujet, change d'état

Solution

Remplacer la dépendance de B vers A par une dépendance sur une interface minimaliste

Conséquences

Couplage abstrait, broadcast, mises à jour en cascade

Catégorie

Structurel

Patron de conception Observer

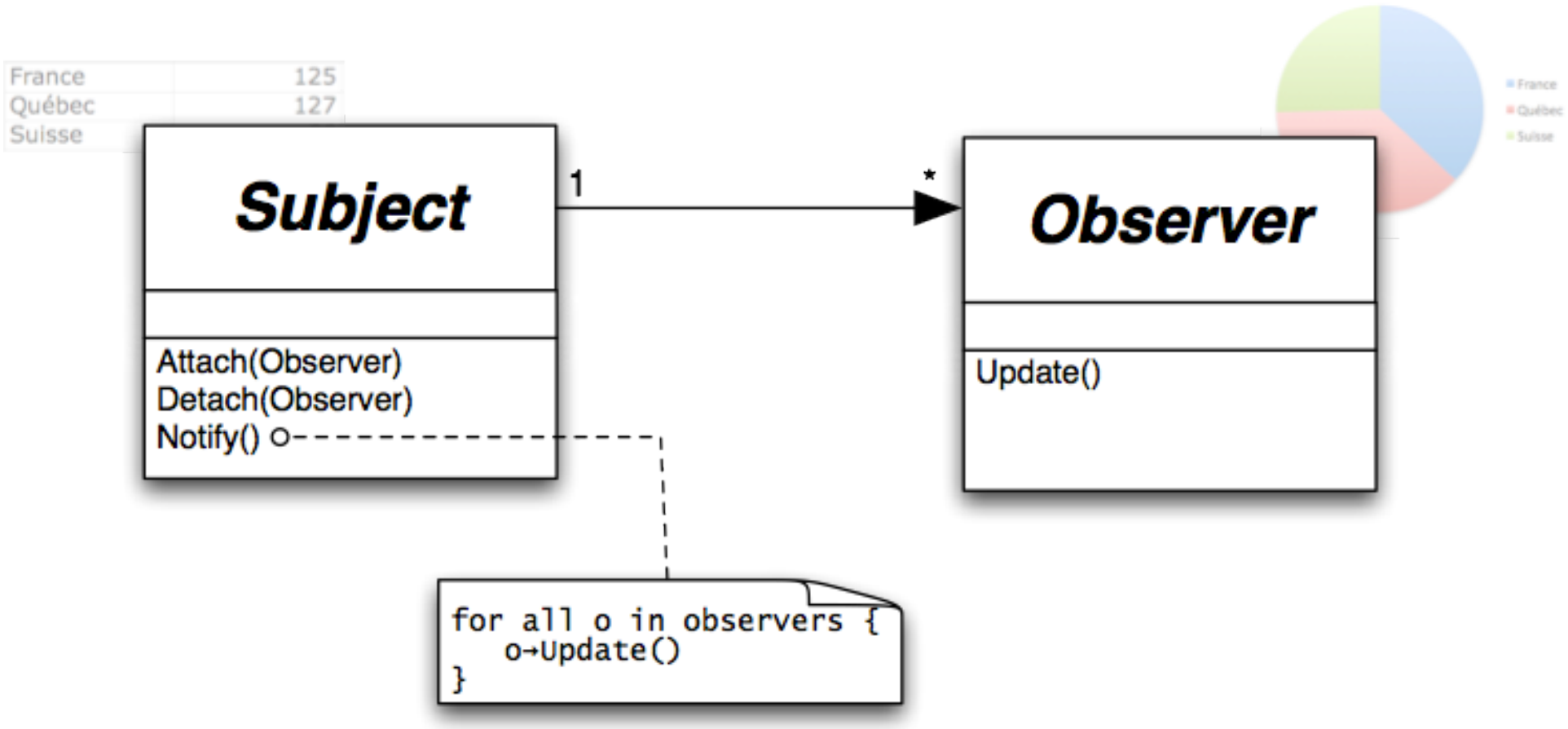
Un *observateur* observe des *observables*.

En cas de **notification**, les observateurs effectuent une action en fonction des informations qui viennent des observables.

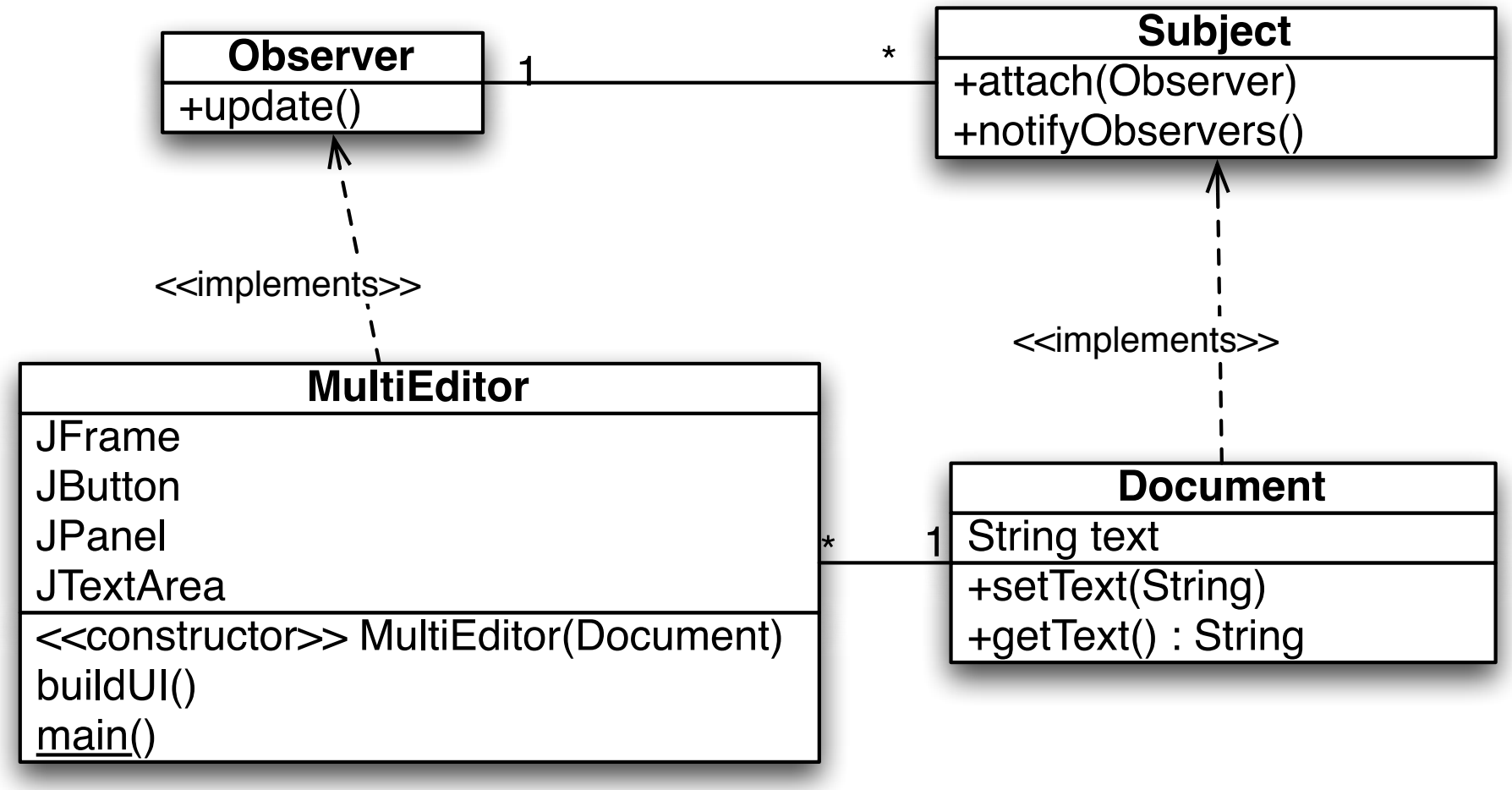
La notion d'**observateur/observable** permet de coupler des modules de façon à réduire les dépendances aux seuls phénomènes observés.

Notions **d'événements (voir cours sur les événements)**

Patron de conception Observer



Patron de conception Observer : exemple



Patron de conception **Command**

Problème

Annuler et/ou sauvegarder des actions sur des objets

Solution

Un objet encapsule toute l'information nécessaire pour appeler une méthode plus tard (nom, propriétaire, paramètres, …)

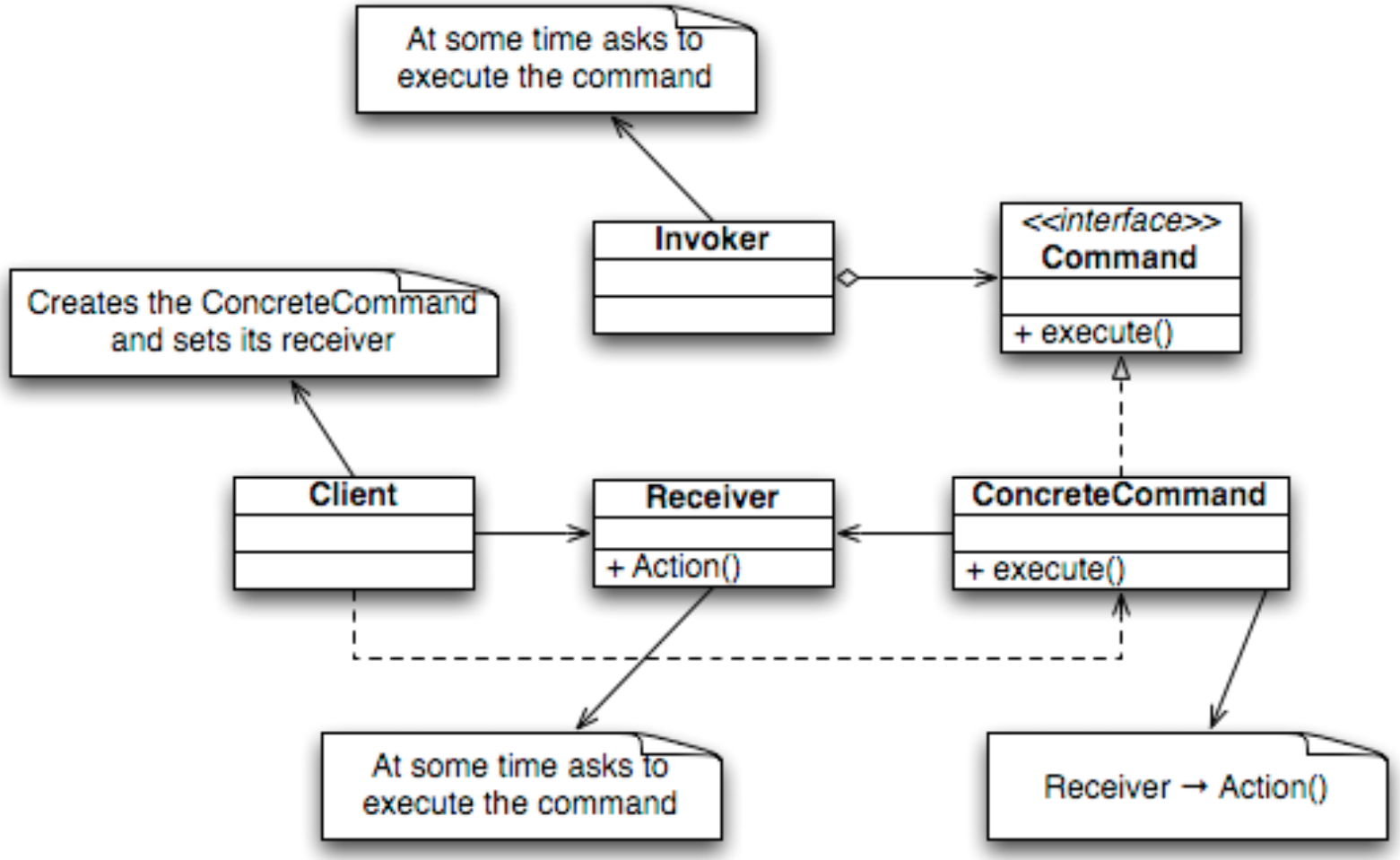
Conséquences

Support de l'historique, de l'annulation de commande, …

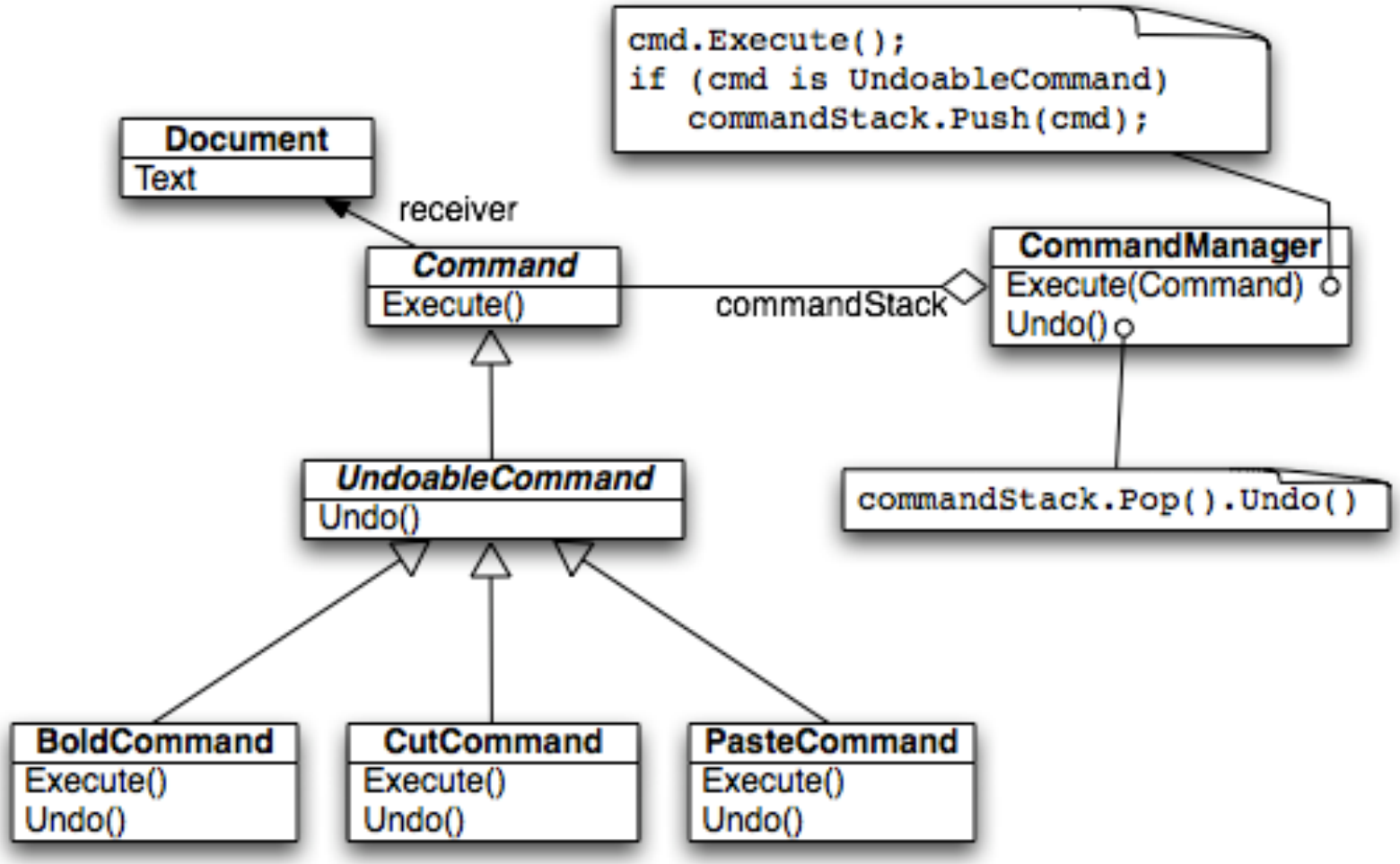
Catégorie

Comportemental

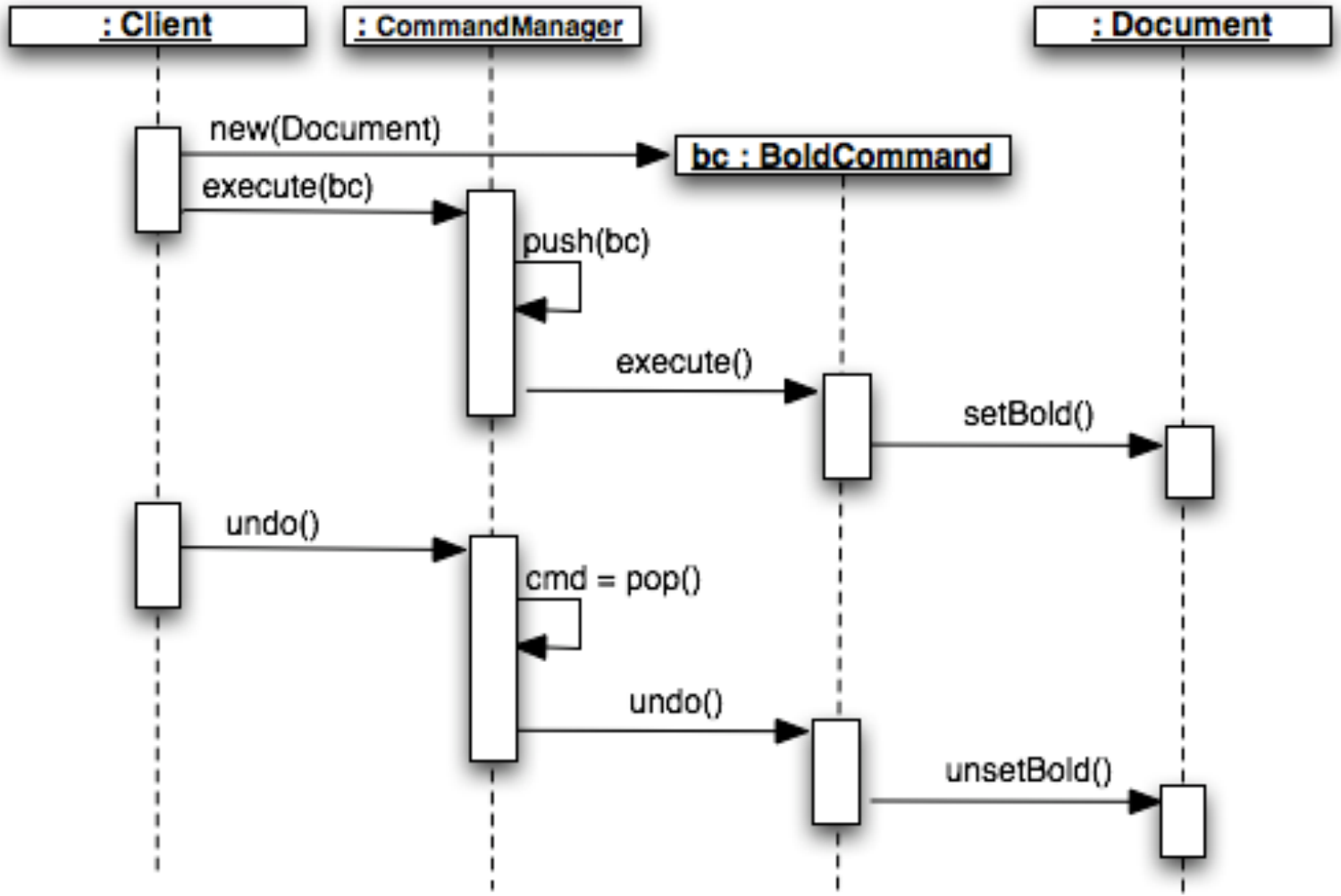
Command Pattern : structure



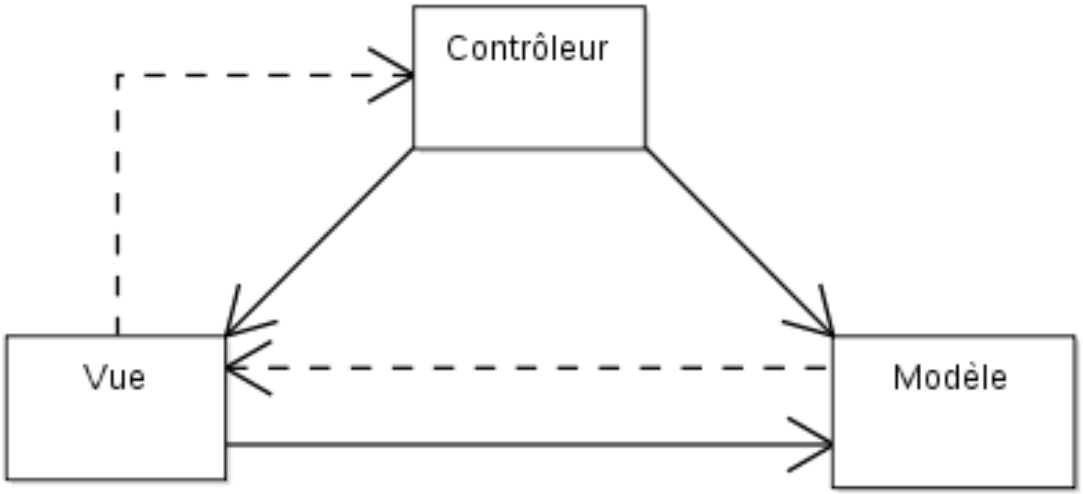
Command Pattern : exemple



Command Pattern : exemple

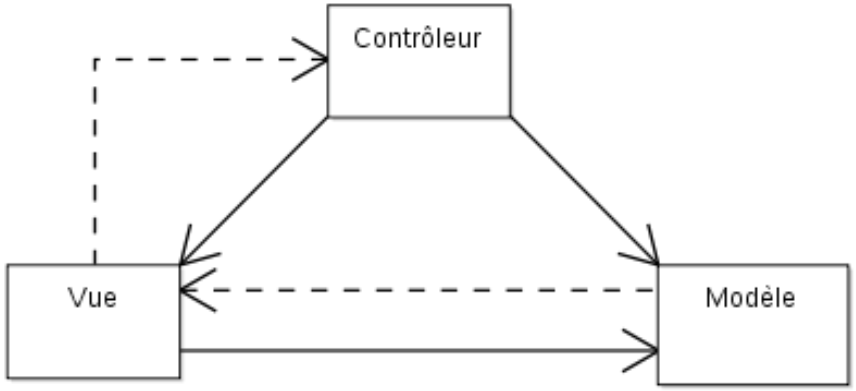


Le patron Model-View-Controller



Model-View-Controller

Pas un pattern en tant que tel mais une méthode de conception pour organiser **l'architecture d'une application avec interface graphique**.



On divise l'application entre :

Modèle (de données, connection avec la DB, etc, notification de la vue)

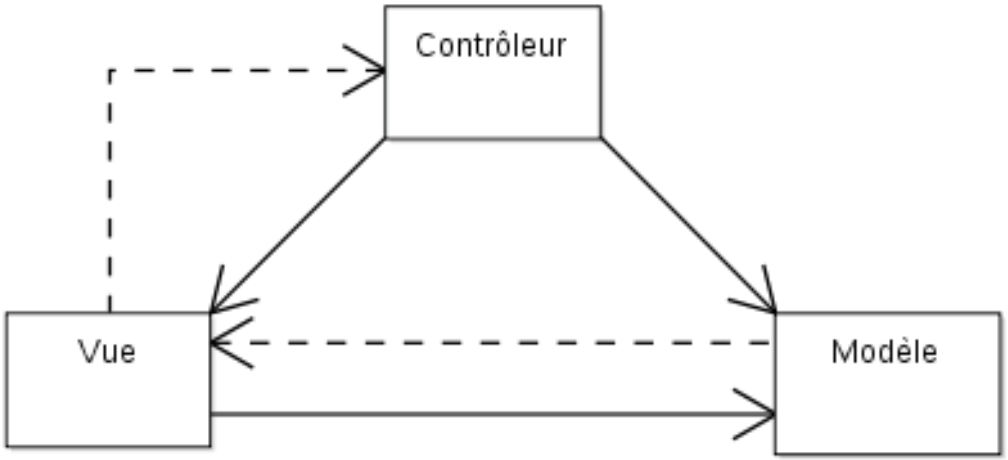
Vue (présentation, interface, notification du controleur, répond aux notifications du modèle)

Contrôleur (logique de contrôle, synchronisation, répond aux notifications de la vue)

Modèle

Comportement de l'application :

- traitement des **données**
- interaction avec la base de données
- garanti l'intégrité des données
- signale à la vue que les données ont changé (via un événement)



Le modèle est le **seul à modifier** les données, il est le garant de ces données.

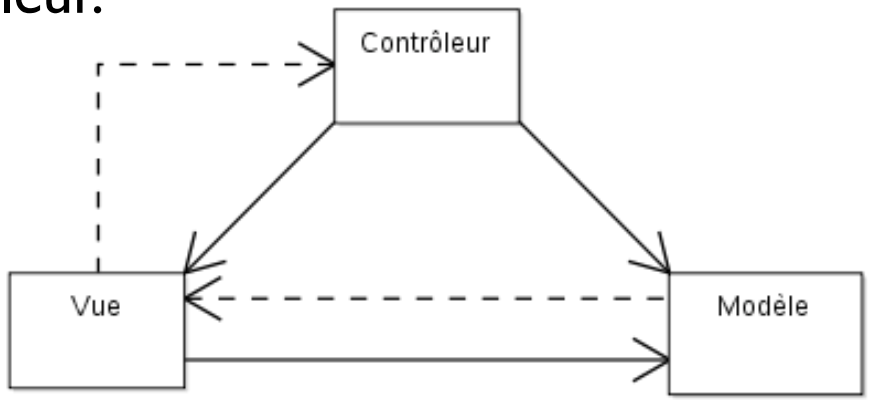
Vue

Interface avec laquelle l'utilisateur interagit

Deux rôles :

Présenter les résultats renvoyés par le modèle

Recevoir les **actions des utilisateurs** (click,, …) et transmettre au contrôleur.
(via un événement)



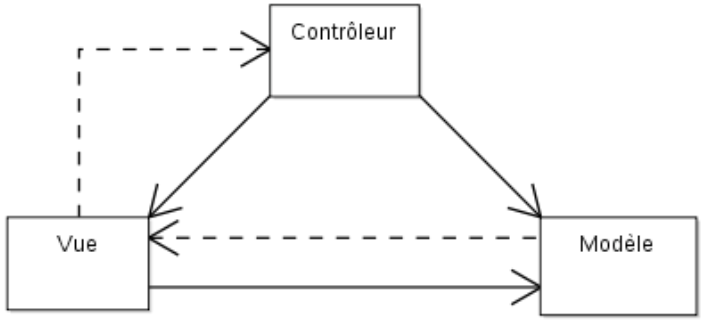
N'effectue **aucun traitement**, ne fait qu'afficher les données du modèle.

Contrôleur

Gestion des évements pour mettre à jour la vue ou le modèle.

Synchronise la vue et le modèle.

Reçoit les notifications de la vue, met à jour le modèle au besoin et avertit la vue que les données ont changé pour que celle-ci se mette à jour.



Le contrôleur ne modifie aucune donnée.

Le contrôleur est composé des méthodes de gestion d'événement.

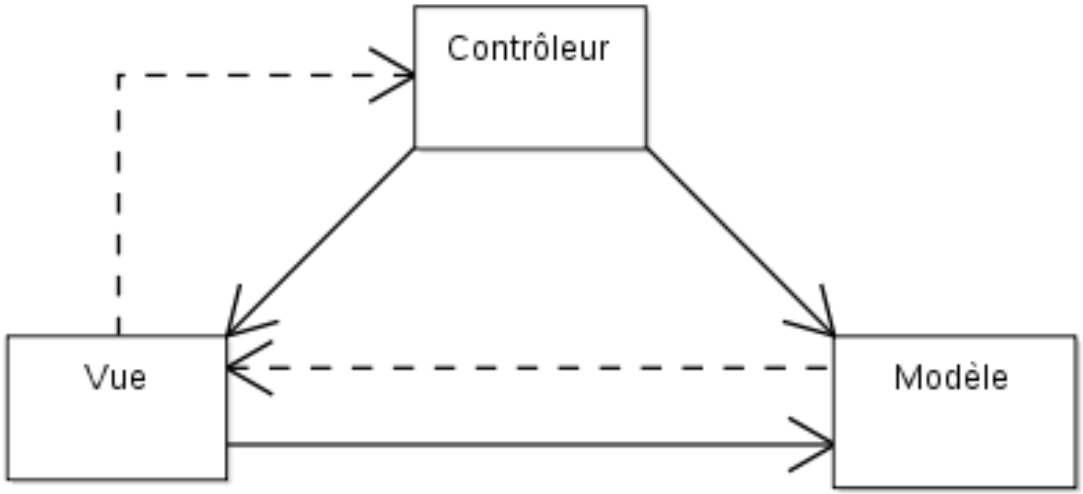
En résumé

L'utilisateur clique sur quelque chose (->notification)

La requête est analysée par le *controleur*

Le *contrôleur* demande au *modèle* de faire ces changements

Le *contrôleur* renvoie la *vue* adaptée, si le *modèle* ne l'a pas déjà fait.



Pointillés = notifications
Plein = association

MVC : Règles de base

Le modèle ne peut pas connaître la vue.

La vue peut connaître le modèle (et l'utilise en lecture seule).

Bibliographie

[1] Larman, Craig (2005) Applying UML and Patterns – An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd ed.). New Jersey: Prentice Hall.

[2] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides (1994) Designs Patterns: Elements of Reusable Object-Oriented Software. AddisonWesley.