Analyse et Conception Orientées Objets

Cours 8 : Introduction aux Design Patterns

Généralités sur les designs patterns

- Un **design pattern** (ou modèle de conception) est un problème de conception récurrent accompagné de sa solution.

- Il est défini par :
  - un **nom** qui l'identifie sans ambiguïté
  - l'**énoncé du problème** récurrent
  - la **formalisation de sa solution** en UML et éventuellement dans tel ou tel langage orienté objet.

Généralités sur les designs patterns (2)

- Parmi les équipes les plus connues dans l'étude des designs patterns, nous avons l'équipe de GoF (gang of four) composée de
  - **Erich Gamma**
  - **Richard Helm**
  - **Ralph Johnson**
  - **John Vlissides**
- Cette semaine, nous regarderons de plus près quelques designs patterns de **GoF**.

Composition d'un Design Pattern

- De son **intention** : ce qu'il fait, son but,
- Sa **motivation** : un scénario qui illustre un cas de conception
- Ses **constituants** : les différentes classes (abstraites et concrètes), interfaces et objets utilisés
- Sa **structure** : sa solution quasiment tout le temps exprimée en UML.
- Ses **collaborations** : comment les instances communiquent entre elles.

Les familles de patterns

- Les créateurs : comment bien créer des objets ?
- Les structuraux : comment bien structurer un ensemble d'objets dépendants ?
- Les comportementaux : comment répartir au mieux les rôles entre les différents objets ?

COO et Les Design Patterns

- COO
  - Programmer avec des interfaces plutôt qu'avec des classes spécifiques
  - Réutilisation : penser plutôt Boite noire (Composition) que boite blanche (héritage)
- Design Patterns
  - Les problèmes sont récurrents s'ils sont exprimés de façons génériques ⇒ Utilisation d'interfaces
  - La bonne solution de beaucoup de problèmes récurrents est pensée en terme de réutilisation ⇒ composition plutôt qu'héritage

Plan de la présentation

- Patterns Comportementaux
  - Commande
  - Observateur

Pattern Commande - Motivation

- On considère une application qui permet de faire des dessins.
  - L'item de menu "nouveau" permet de créer et d'ajouter à l'application un nouveau dessin
  - L'item de menu "coller" permet d'insérer dans un dessin le contenu du presse-papier

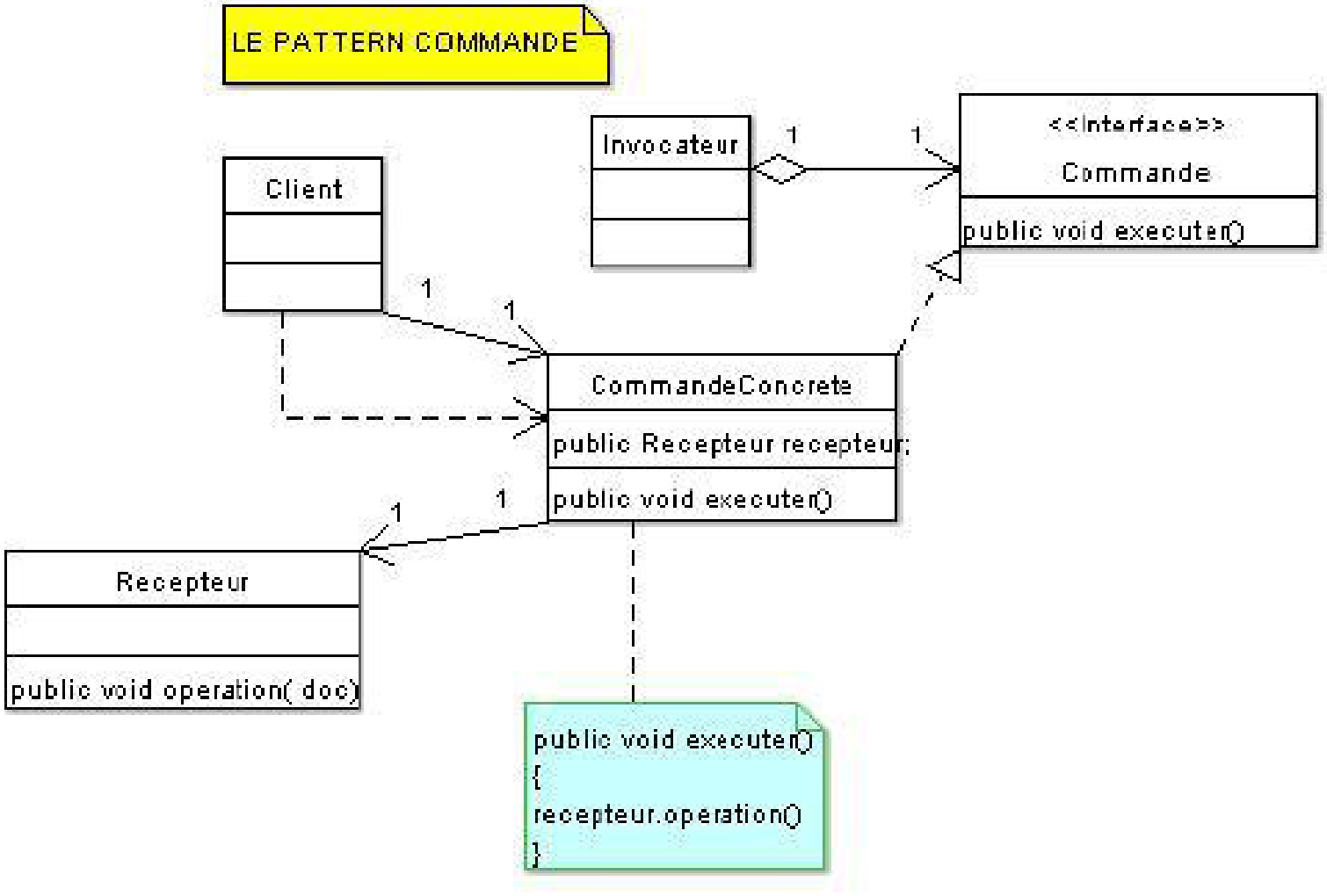
Pattern Commande - Motivation

- On remarque que les deux objets "Item" ont à transmettre une requête à des objets totalement différents :
  - Un objet Application pour le premier
  - Un objet Dessin pour le second.
- Pourtant ces deux objets ont en commun d'être des items de menu donc issus de la même classe.

Pattern Commande - Intention

- Nous voulons :
  - encapsuler une requête comme un objet pour découpler complètement l'objet qui émet la requête (émetteur) de celui qui va la traiter. (Récepteur)
  - l'objet émetteur n'a aucune information sur la nature du traitement de la requête.
  - Le traitement de la requête n'est plus exécuté directement par l'objet concerné (émetteur)

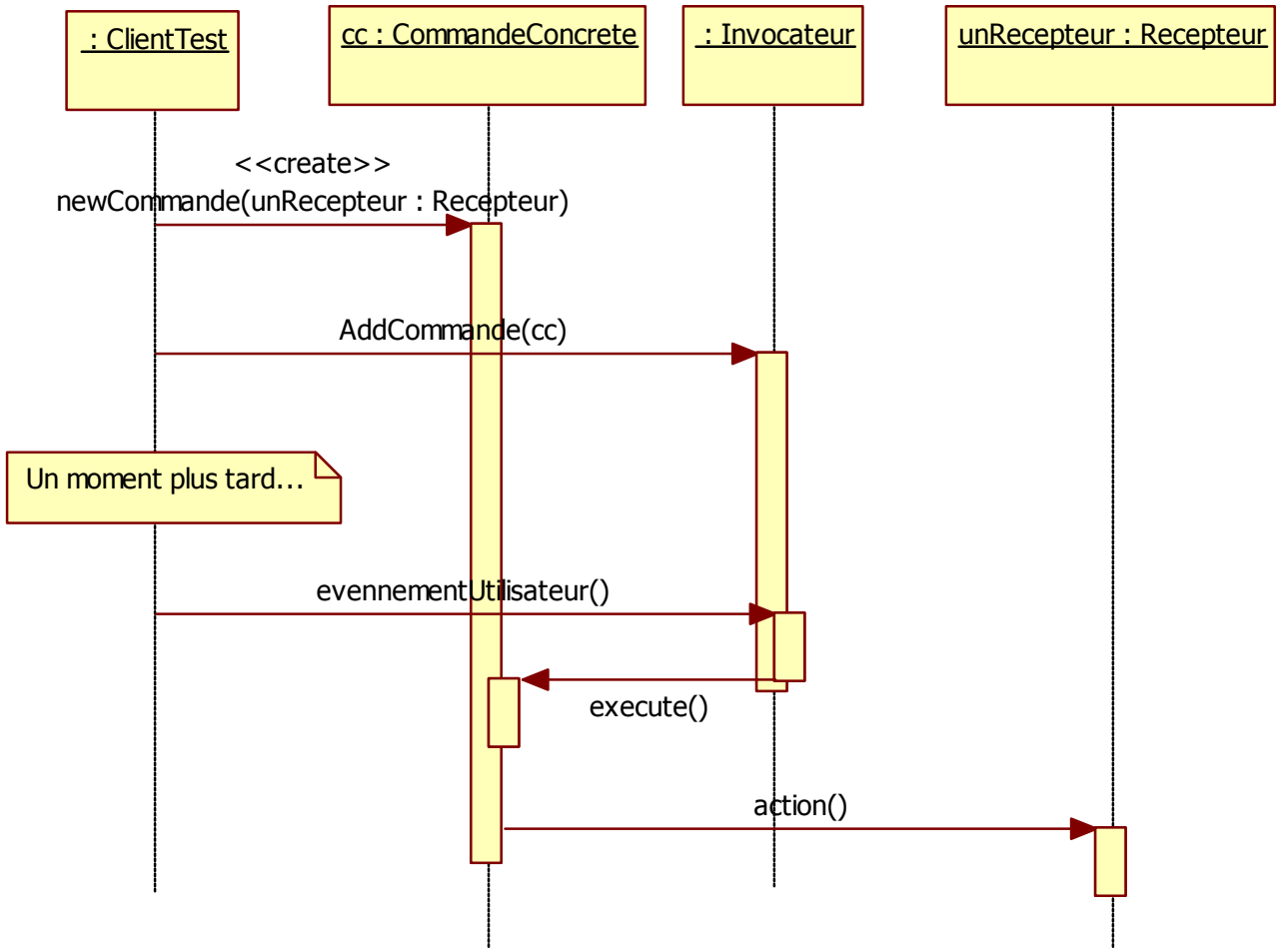
Pattern Commande - Structure



Pattern Commande - Constituants

- Une interface **Commande** pour exécuter une opération
- Une ou plusieurs classes **Commandes concrètes** (CmdColler, CmdNouveau) qui implémentent Commande.
- Une ou plusieurs classes **Invocateur** (les items de menus)
- Des classes **Récepteur** (Application, Document) dont les objets sont impactés par l'exécution de la commande. Ses objets exécutent la commande.
- Une classe **Client** (Application) qui crée la commande concrète, le récepteur et le positionnement du récepteur dans la commande concrète

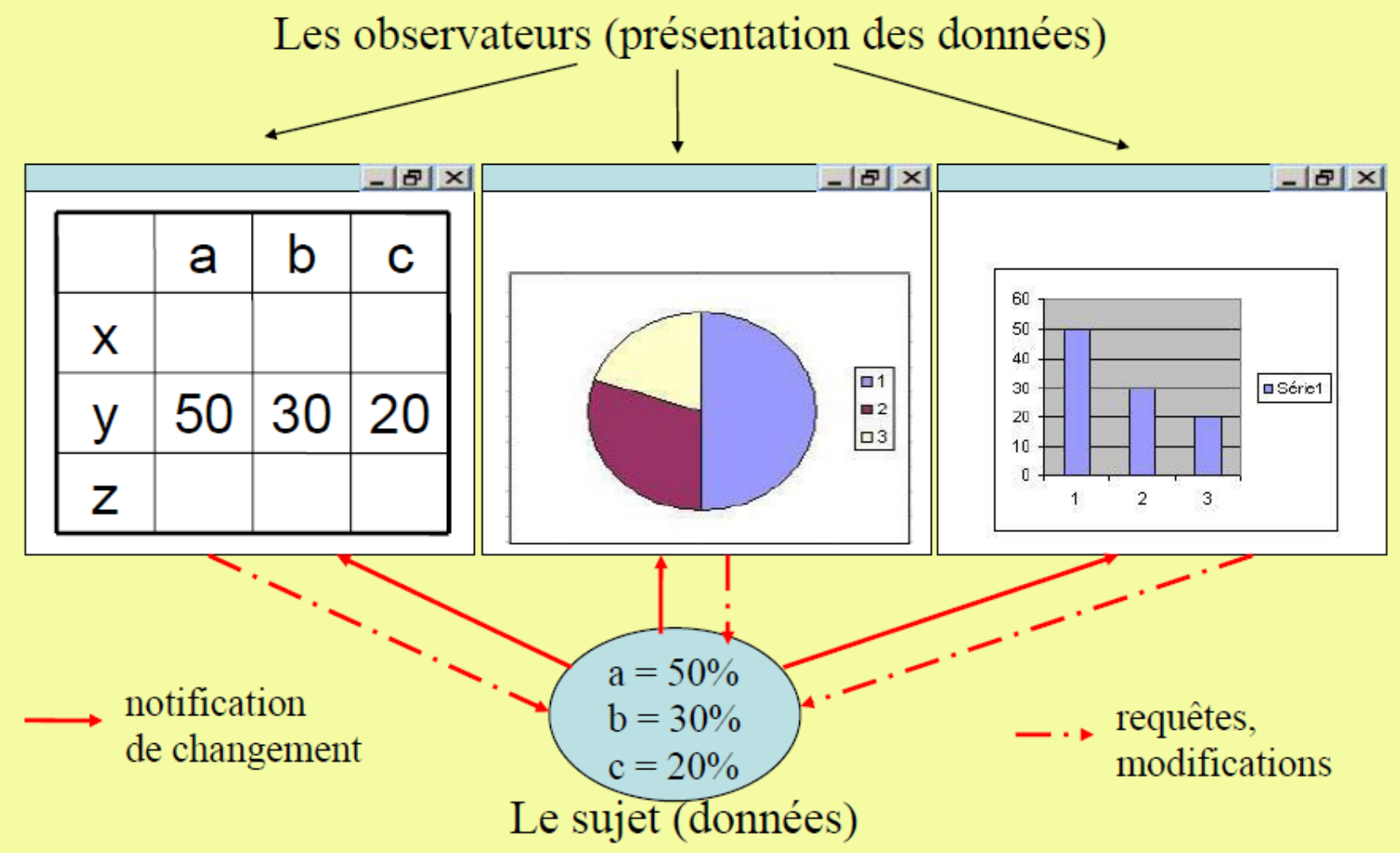
Pattern Commande - Séquence



Pattern Observateur - Motivation

- Dans de nombreuses IHM, les mêmes données peuvent être présentées à l'utilisateur de plusieurs façons différentes.
- Comment rendre les données et leurs représentations indépendantes ?

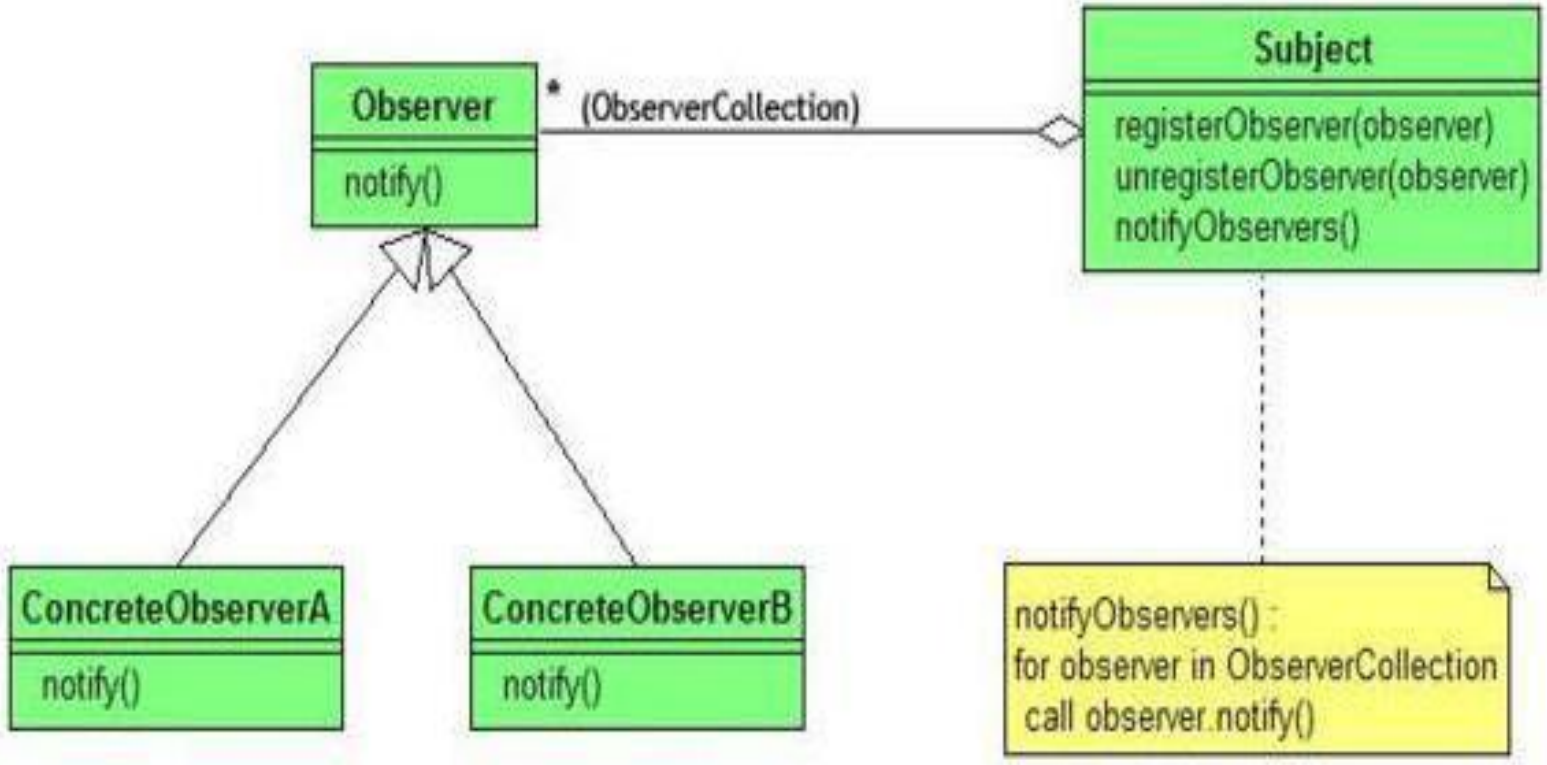
Pattern Observateur - Motivation



Pattern Observateur - Intention

- Nous voulons :
  - Créer des dépendances entre un objet X (les données) et ses objets liés (les affichages) de telle façon que dès que l'objet X change d'état tous les objets liés en soient notifiés automatiquement

Pattern Observateur - Structure



Ce schéma a été obtenu à l'adresse http://en.wikipedia.org/wiki/Observer_pattern

Pattern Observateur - Constituants

- La classe **Subject**
  - connait ses **Observers**
  - fournit une interface pour en permettre l'ajout et la suppression
  - stocke les données qui intéressent les **Observers**
  - envoie une notification à ses **Observers** lorsque ses données changent
- **Observer** définit une interface pour la mise à jour des objets qui doivent être notifiés des changements de l'objet **Subject**
- Une ou plusieurs classes **ConcreteObservers** qui implémentent l'interface Observer.
- Une classe **Client** (Application) qui crée le **Subject**, les **ConcreteObservers** et les associent.

Pattern Observateur - Séquence