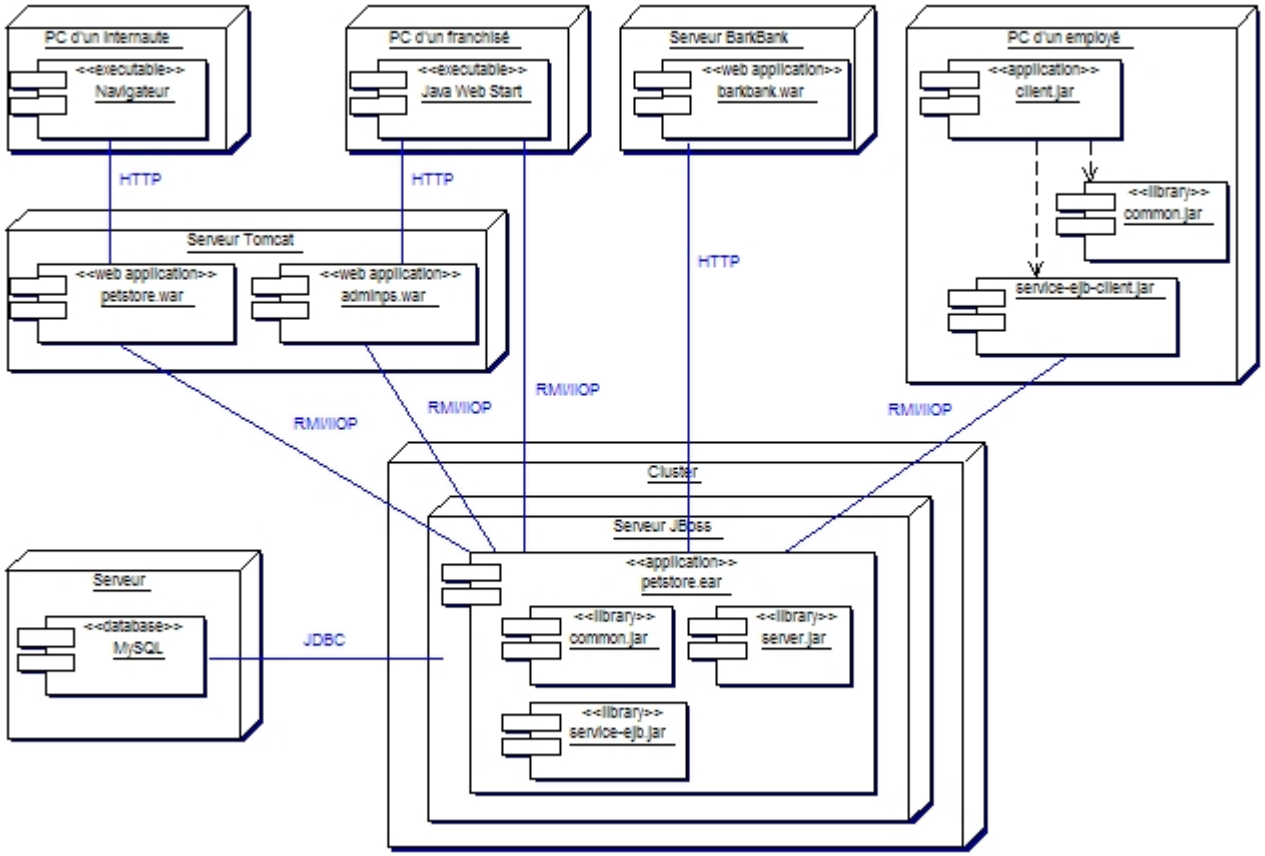
Les diagrammes en pratique

- **Les diagrammes de déploiement**
  - **Vue physique du système**
  - **Fichiers + Hardware + Réseau**
  - **Machine : node avec stéréotype <<device>>**
  - **Fichier : composant <<artifact>>**
  - **Application Server (conteneur) : node avec stéréotype <<executionEnvironment>>**
  - **Association entre nœuds : communication (IP…)**

Les diagrammes en pratique

- **Les diagrammes de déploiement**

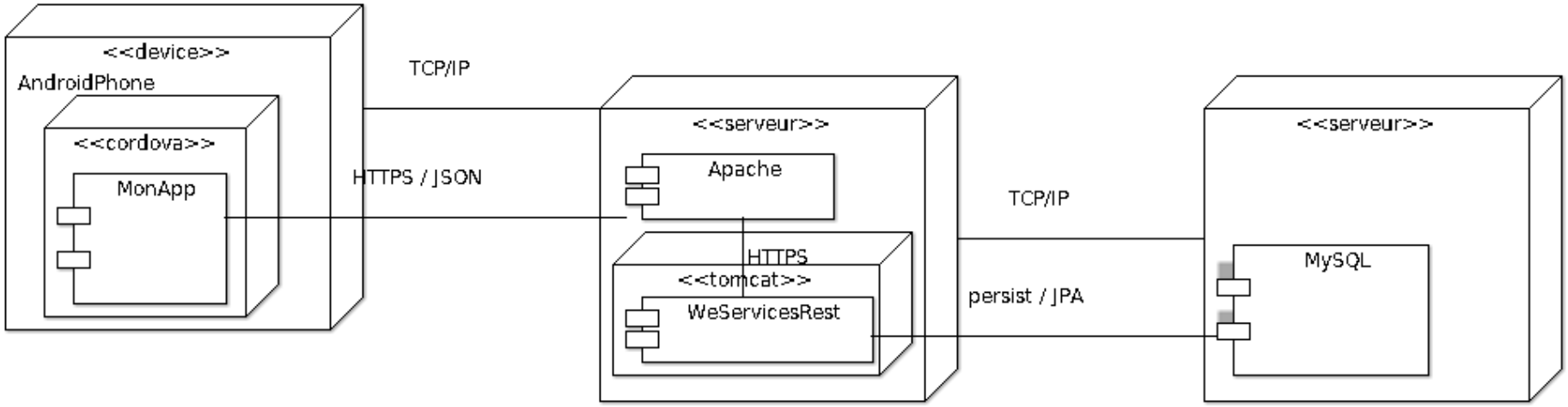


Les diagrammes en pratique

- **TP Diagramme de déploiement :**
  - **Vous devez modéliser le déploiement d'une application mobile ayant des interactions avec un serveur web par web services REST. Ce webserver (Apache) et relié au conteneur de Servlet Tomcat. L'application Java utilise la persistance objet et les données sont stockées sur un autre serveur à l'aide de MySQL.**

Les diagrammes en pratique

- **TP Diagramme de déploiement : un corrigé**

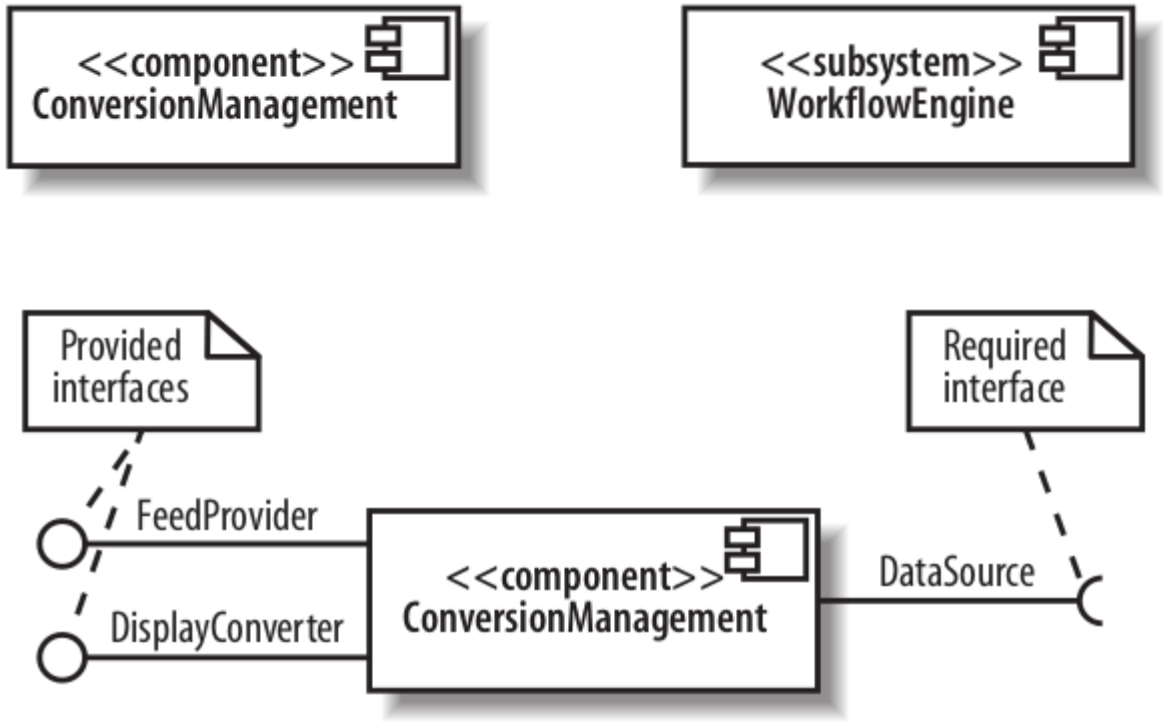


Les diagrammes en pratique

- **Diagramme de Composants**
  - **Lorsque définir directement les classes est trop complexe**
  - **Permet un niveau d'abstraction plus élevé que le diagramme de classes**
  - **On modélise des « grosses briques »**
    - **Composant = ensemble fonctionnel de classes encapsulées réutilisable**

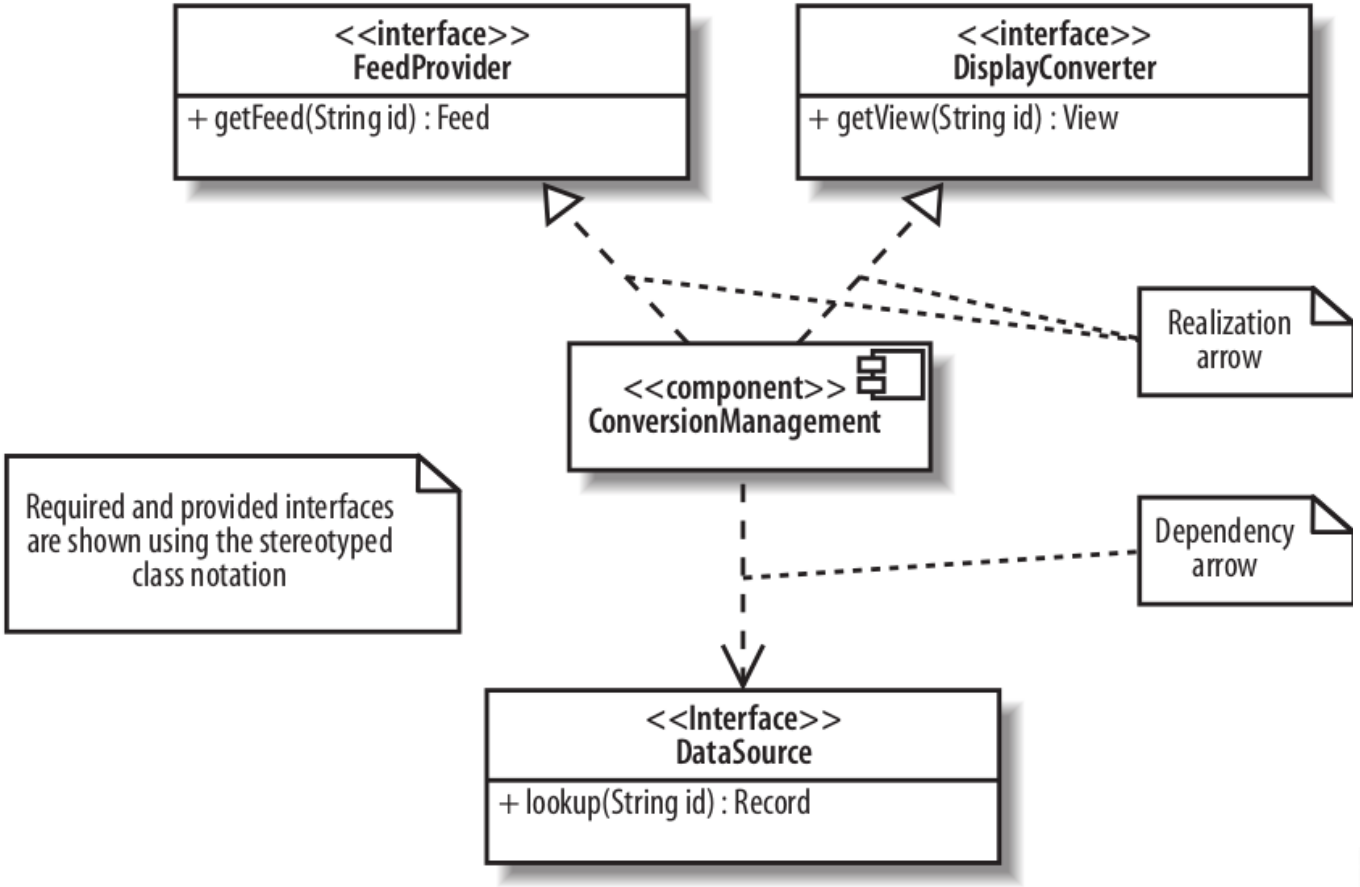
Les diagrammes en pratique

- **Diagramme de composants : les éléments**



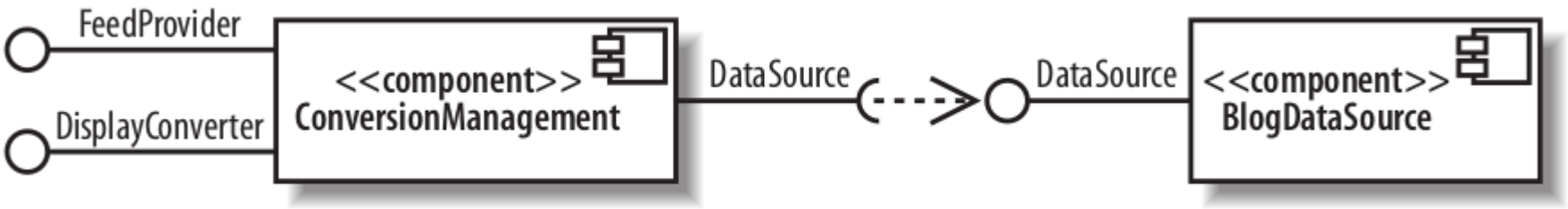
Les diagrammes en pratique

- **Diagramme de composants : les éléments**



Les diagrammes en pratique

- **Diagramme de composants : les éléments**
  - **Deux composants fonctionnant ensemble**

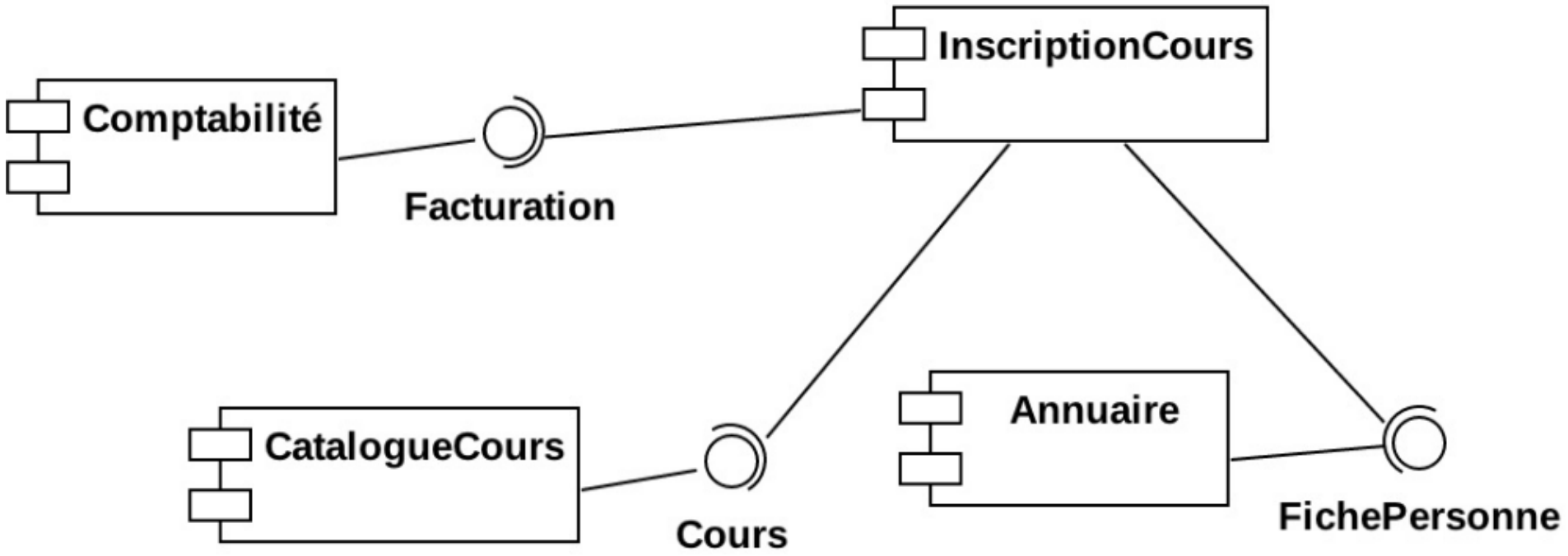


Les diagrammes en pratique

- **TP Diagramme de composants**
  - **Sur papier faire un diagramme de composants représentant les relations entre l'outil d'inscription aux cours, le système de comptabilité (pour la facturation), le catalogue de cours (qui fourni le cours au système), l'annuaire de personnes**

Les diagrammes en pratique

- **TP Diagramme de composant : un corrigé**



Designs patterns et UML

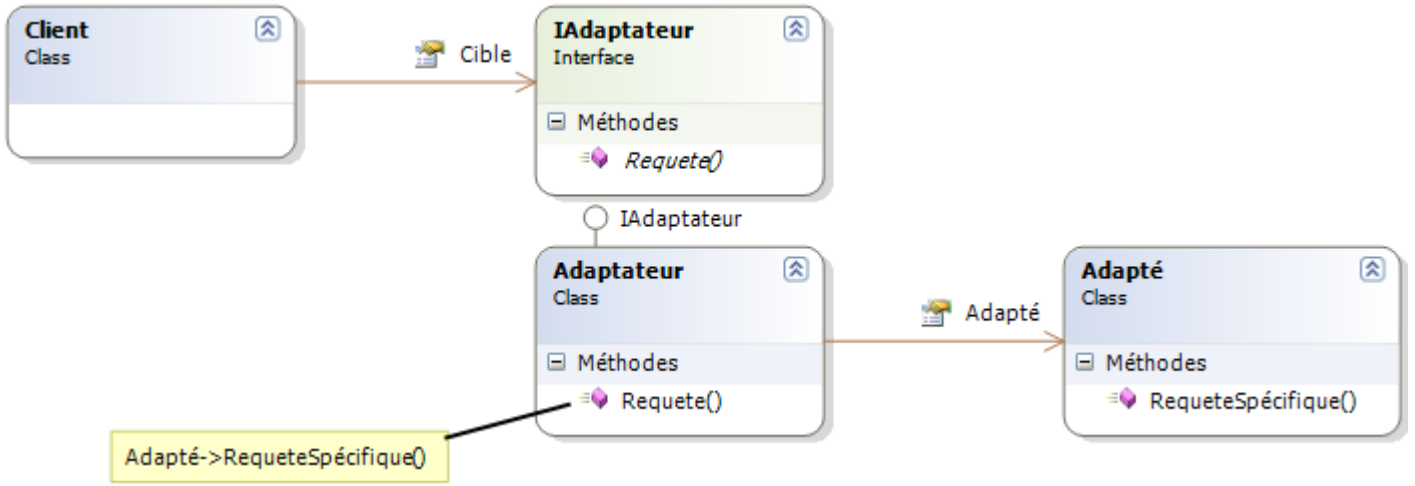
- **Design patterns → La suite : les designs patterns de structure**
  - **Bonnes pratiques de conception en ce qui concerne l'héritage et les interfaces**

Designs patterns et UML

- **Design pattern structural : Adapter (aka Wrapper)**
  - **Permet de traduire l'interface d'une classe en une autre**
  - **Quand on veut utiliser une API ou une classe dont les fonctionnalités sont là mais pas les bonnes signatures de méthode**
  - **Quand on veut ajouter/changer un fonctionnement à une classe existante**

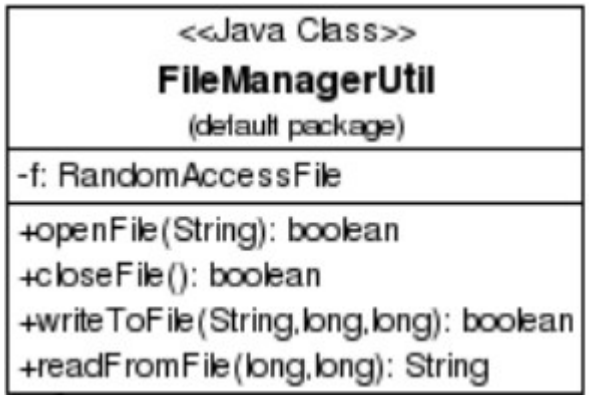
Designs patterns et UML

- **Design pattern structural : Adapter (aka Wrapper)**



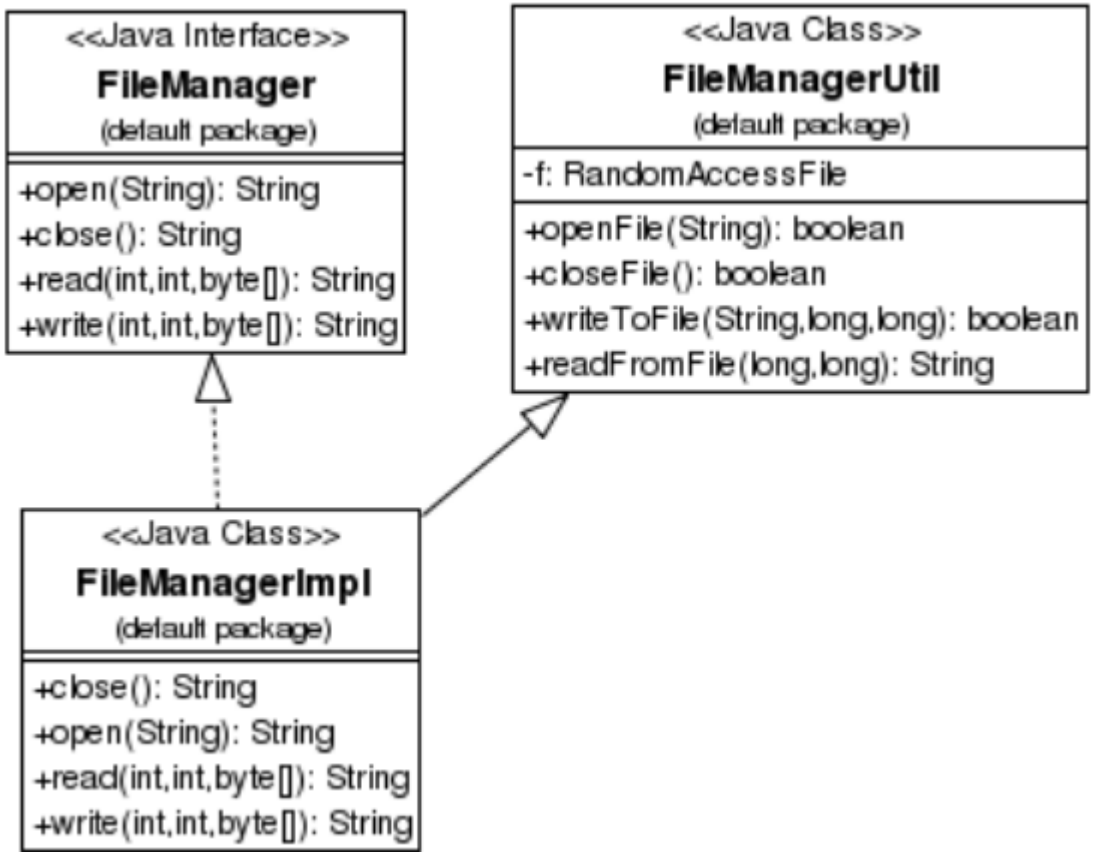
Designs patterns et UML

- **TP Adapter :**
  - **Modélisez le diagramme de classe représentant l'adaptation de la classe FileManagerUtil pour répondre à l'interface FileManager**



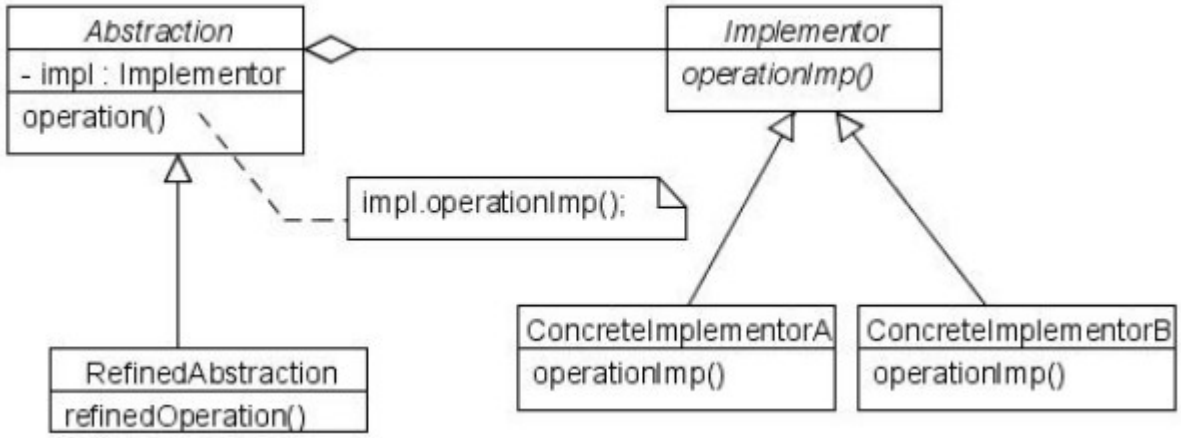
Designs patterns et UML

- **TP Adapter : un corrigé**



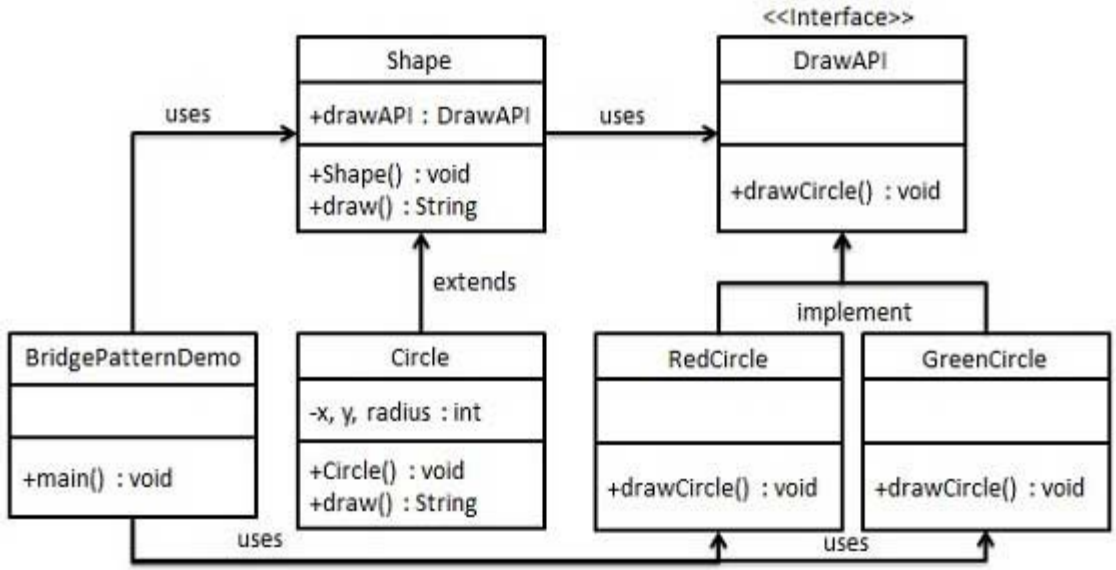
Designs patterns et UML

- **Design pattern structural : Bridge**
  - **Permet de complètement découpler deux implémentations : le couplage à lieu au niveau des interfaces, les implémentations peuvent varier indépendamment**



Designs patterns et UML

- **Design pattern structural : Bridge**

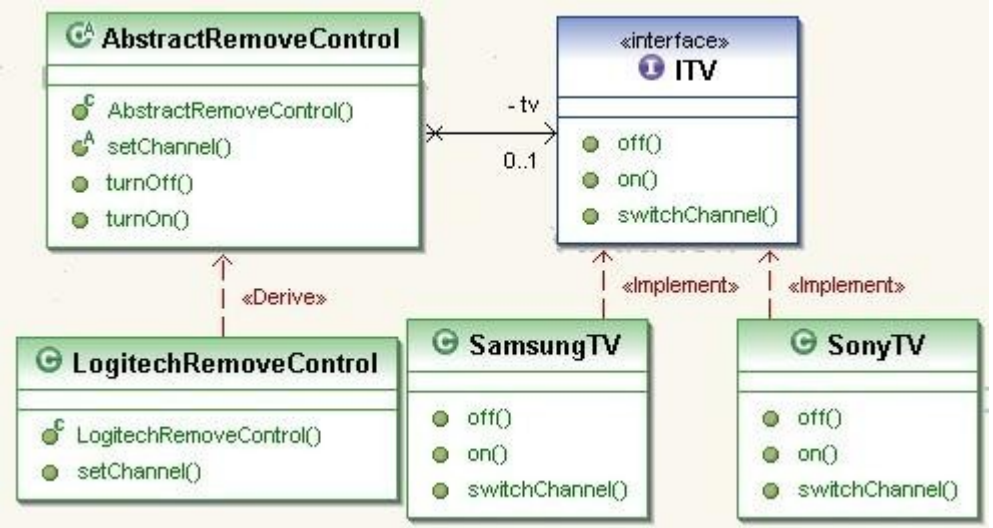


Designs patterns et UML

- **TP Bridge :**
  - **Modéliser en utilisant le design pattern bridge, l'utilisation d'une TV par une télécommande. On imaginera deux modèles concrets de TV et un modèle concret de télécommande.**

Designs patterns et UML

- **TP Bridge : un corrigé**

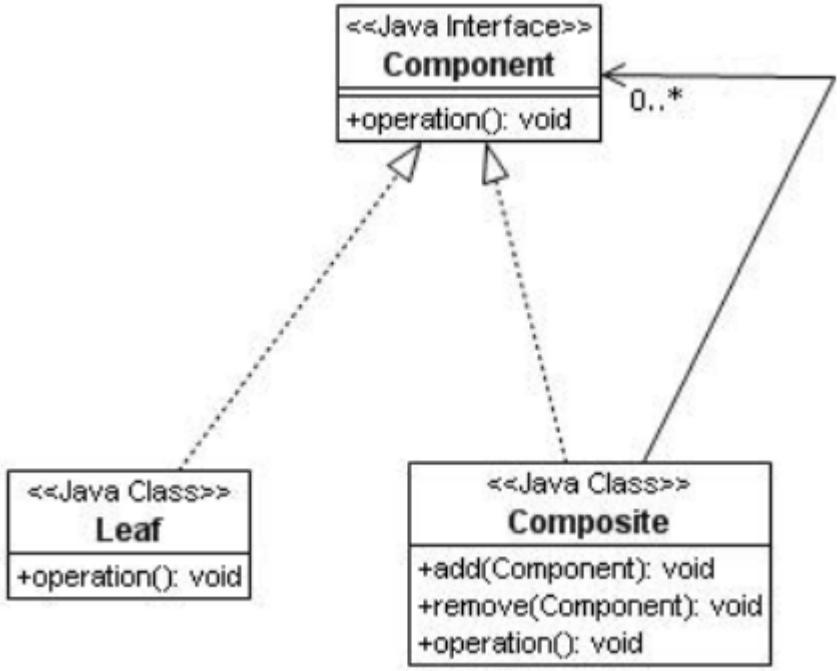


Designs patterns et UML

- **Design pattern structural : Composite**
  - **Permet de modéliser une relation arborescente de composants sans se préoccuper du fait d'être sur un nœud ou une feuille**
  - **Utile si la structure est très complexe et change régulièrement (ajout, suppression d'éléments)**
  - **On utilise pleinement le principe d'encapsulation (un ensemble de composants est un composant…)**

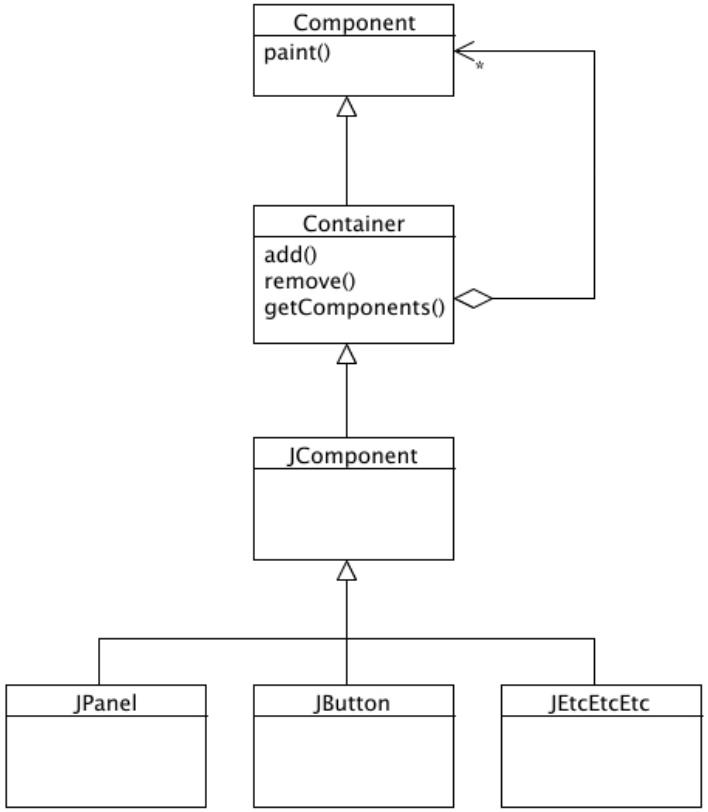
Designs patterns et UML

- **Design pattern structural : Composite**



Designs patterns et UML

- **Design pattern structural : Composite**

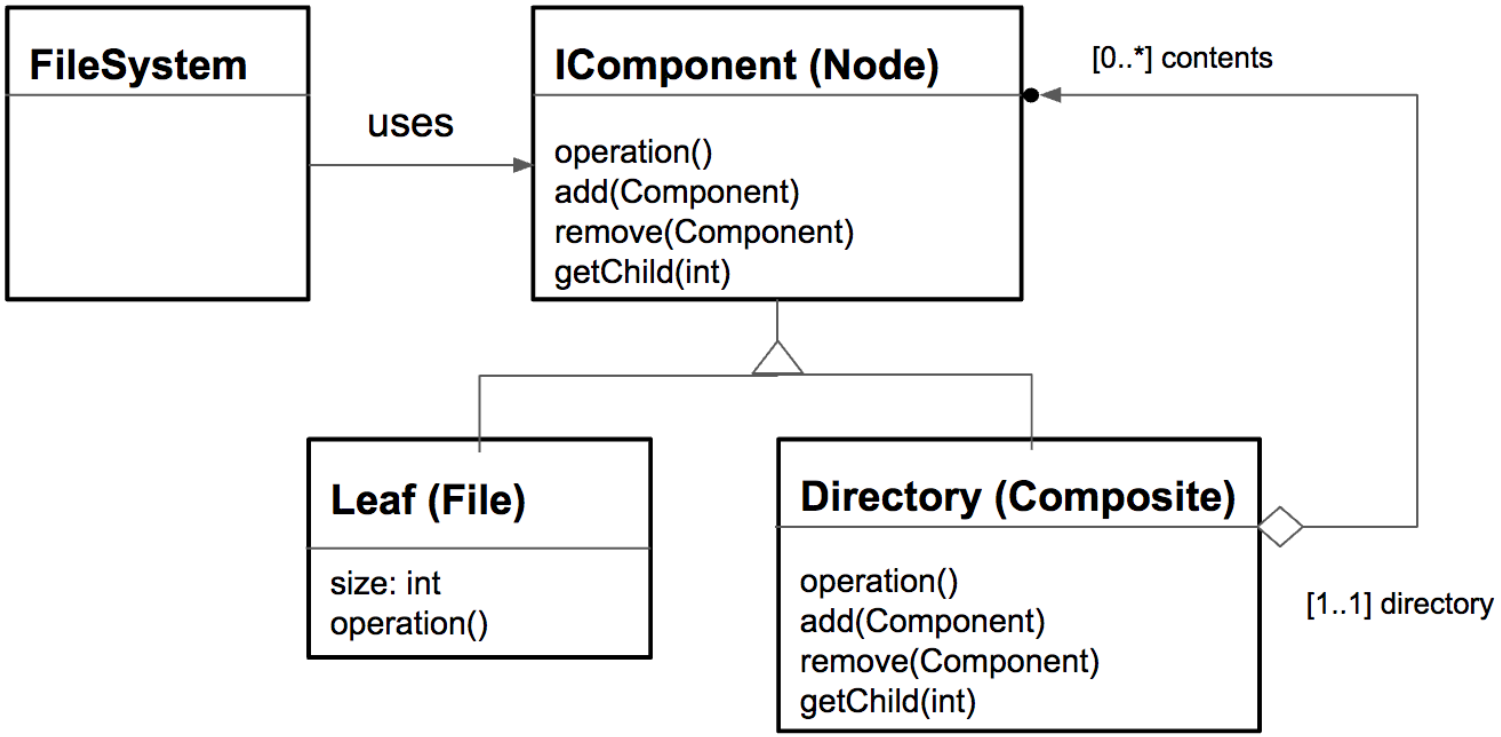


Designs patterns et UML

- **TP Composite :**
  - **Modéliser sous la forme d'un diagramme de classe en utilisant le composite, les fichiers et répertoires d'un système de fichiers.**

Designs patterns et UML

- **TP Composite :**

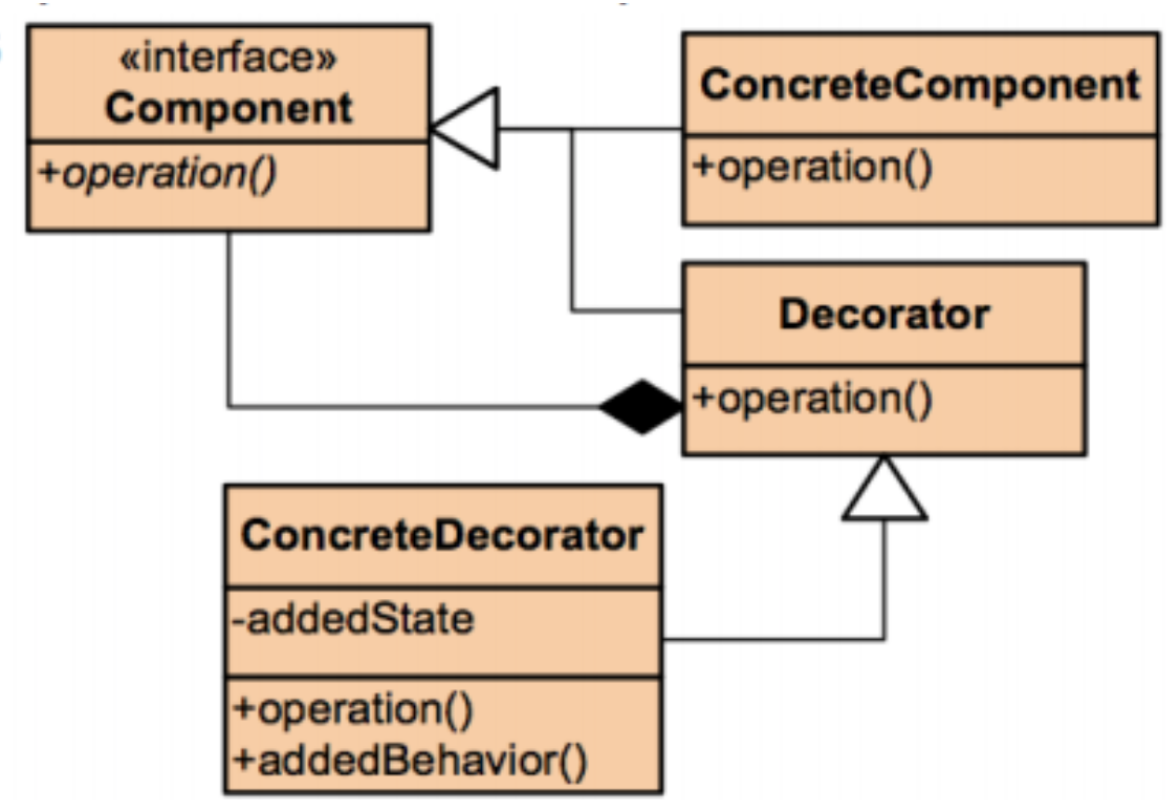


Designs patterns et UML

- **Design pattern structural : Decorator**
  - **Permet d'ajouter des fonctionnalités sur une classe sans modifier celle-ci en utilisant la composition**
  - **Permet de modifier l'objet pendant le runtime**
    - **Voiture v = new Clio() ;**
    - **v = new SuperClio(v) ;**

Designs patterns et UML

- **Design pattern structural : Decorator**

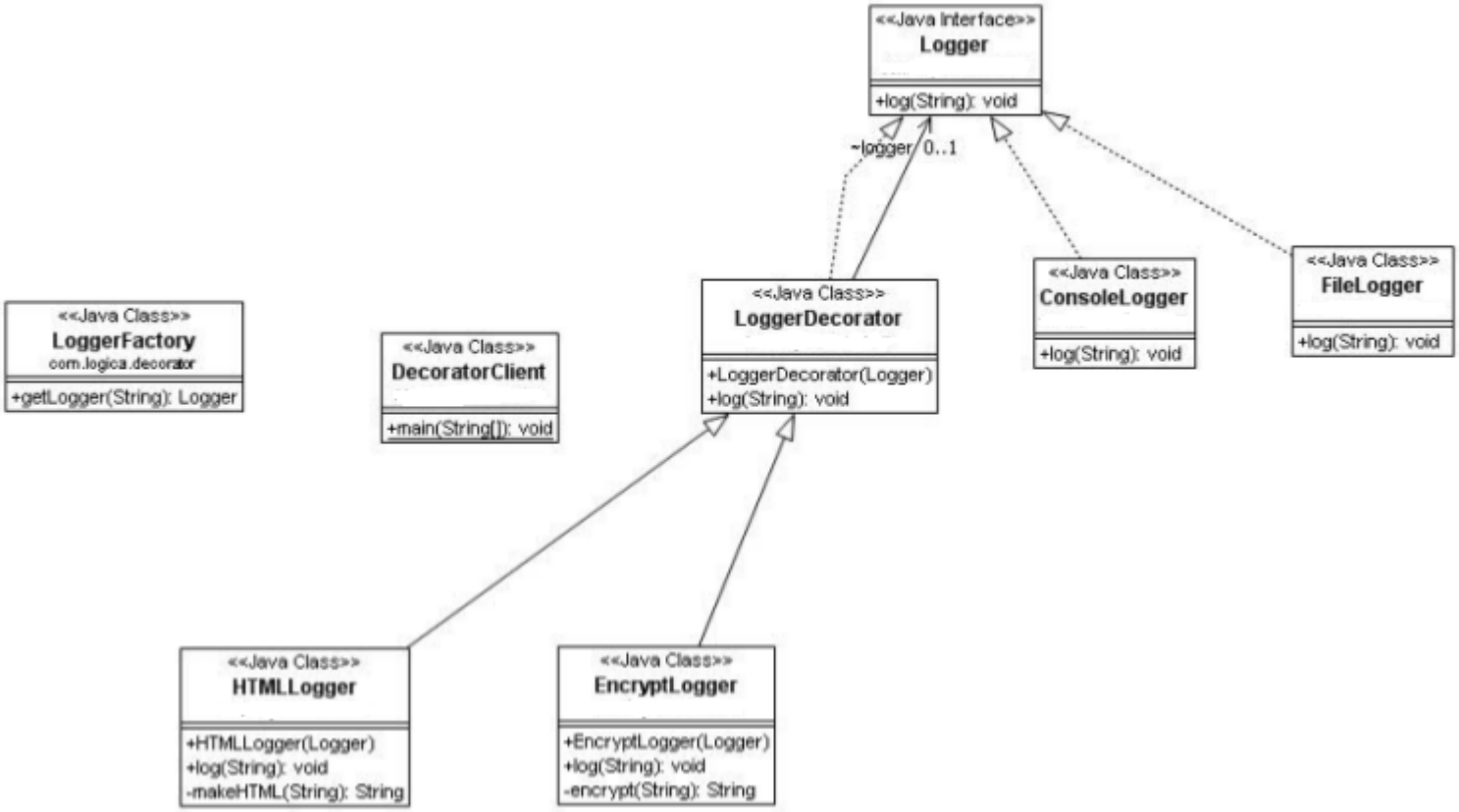


Designs patterns et UML

- **TP Décorator :**
  - **Représenter le diagramme de classe utilisant le decorator d'un système de logging utilisant soit la console, soit un fichier, et pouvant éventuellement décorer le texte du message soit en html, soit en le cryptant.**

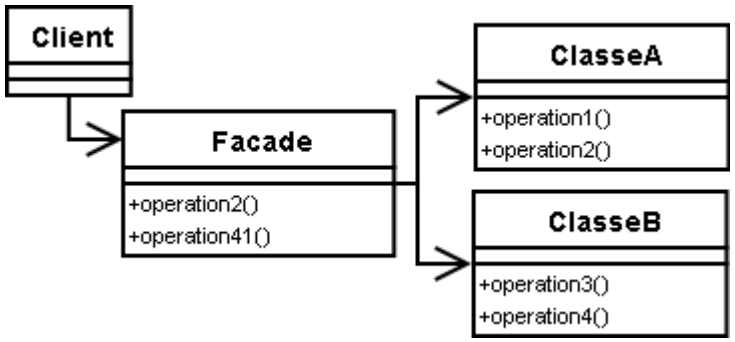
Designs patterns et UML

- **TP Décorator : un corrigé**



Designs patterns et UML

- **Design pattern structural : Facade**
  - **Permet de fournir une interface unifiée d'un système complexe**
  - **Permet de regrouper de multiple petites implémentations ou librairies en un API**



Designs patterns et UML

- **TP Facade :**
  - **Représenter le diagramme de classe de la gestion d'un compte bancaire BankAccountFacade (création de compte, fermeture, dépôt, retrait) en utilisant les classes AccountManager, BankAccount, SecurityManager, AccountReference, AccountPermission,…**

Designs patterns et UML

- **TP Facade : un corrigé**