Vous avez le contrôle, ne soyez ni «fashion», ni «archaïque»

(1) toutes les décisions en matière de conception, qu'elles relèvent de la conception logicielle ou architecturale, doivent être prises à la lumière des contraintes fonctionnelles, comportementales et sociales, et non selon des tendances aléatoires ;
(2) lorsque vous ne maîtrisez qu'une façon de faire, tout à tendance à se ressembler.

« Il est tentant, si le seul outil que vous avez est un marteau, de traiter tout problème comme si c'était un clou. »

—Abraham Maslow, « The Psychology of Science »

Guide de la conception REST et API -- Octobre 2015

Design Patterns

M. Blay-Fornarino & S. Urli

Motivations ... (toujours les mêmes)

- Faire une conception et un développement de qualité :
  - Extensibilité
  - Flexibilité
  - Maintenabilité
  - Réutilisabilité
  - Clarté

Toujours pas de recette «magique»

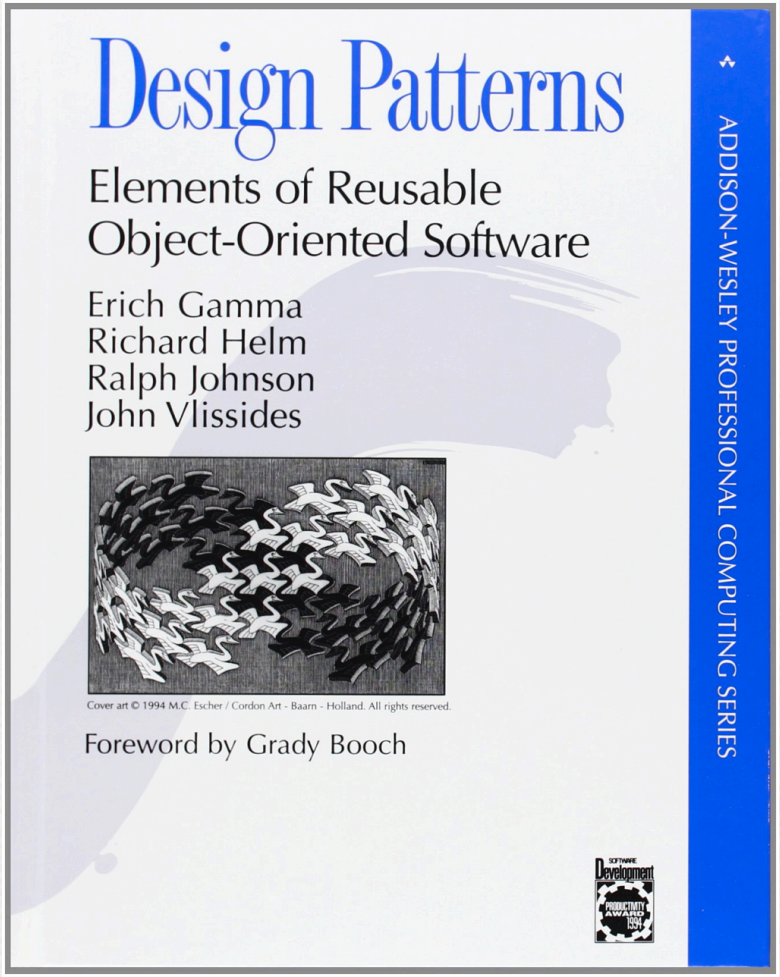
... mais outre les bonnes pratiques :

- KIS : Keep It Simple
- DRY : Don't Repeat Yourself
- YAGNI : You Ain't Gonna Need It
- SOLID

Patron de conception : un outil supplémentaire

Patrons de conception?

- “ Généralisation d’une solution à un problème de conception récurrent par la description des classes et objets communicants ”



Patrons de conception?
Comment sont-ils définis?

- Identification d'un problème de conception récurrent
  - Schématisation du problème de manière générique
  - Description d'une solution sous la forme d'un patron

Patrons de conception?
Comment les utiliser?

- Identifier un problème dont le motif a fait l'objet d'une solution
- Rechercher le patron de conception adapté
- Appliquer et adapter la solution proposée par le patron de conception

Historique

- 1977/79 : Christopher Alexander - Patron de conception pour l'architecture des villes et bâtiments
- 1987 : Kent Beck et ard Cunningham - Papier à OOPSLA sur l'utilisation de patrons de conception pour la programmation orientée objet
- 1994 : le «Gang of Four» (GoF) (Gamma, Helm, Johnson and Vlissides) publie Design Patterns: Elements of Reusable Object-Oriented Software - Présentation des 23 patterns fondamentaux

Classification des patrons

- Patrons de **création** :
  - dédiés à la création des objets.
  - visent l'indépendance entre création et utilisation des objets.
- Patrons de **structure** :
  - dédiés à la composition des objets.
  - visent à conserver une bonne séparation des préoccupations.
- Patrons de **comportement** :
  - dédiés à la communication entre les objets
  - visent à conserver de la flexibilité dans les liens de communication.

Classification

- 23 Patrons fondamentaux
  - Patrons de création :
    - ***Abstract Factory***, Builder, Factory method,
    - Prototype, Singleton
  - Patrons de structure :
    - Adapter, Bridge, ***Composite***, Decorator, Facade, Flyweight, Proxy
  - Patrons de comportement :
    - Chain-of-responsibility, Command, Interpreter, Iterator, Mediator, Memento, ***Observer***, State, Strategy, Template Method, Visitor

Autour des patrons de conception

- Patrons GRASP (General Responsibility Assignment Software Patterns)
- Principes SOLID
- Patrons d'architecture (ex : MVC, layers, etc)
- Patrons de gestion de la concurrence (pool de threads, etc)

Sources

- Les design patterns en Java - Steven John Metsker, William C. Wake
- Head first design patterns - Eric et Elisabeth Freeman
- The design pattern Smalltalk Companion
- Sherman R. Alpert, Kyle Brown, Bobby Woolf
- http://blog.codinghorror.com
- http://en.wikipedia.org/wiki/Software_design_pattern

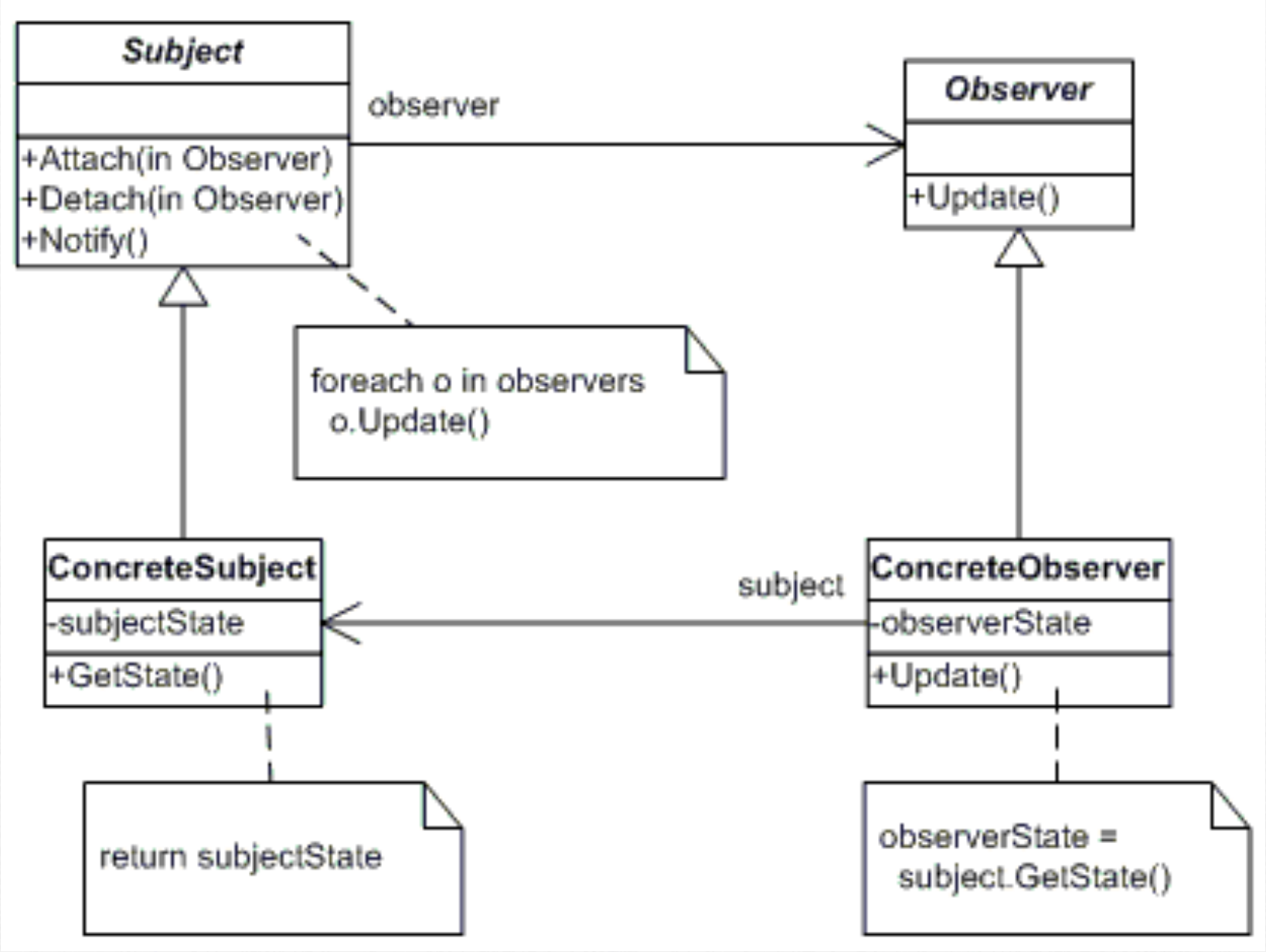
Design Pattern Observer

«Pattern Observer» : le problème

- ✓Problème : Mettre en oeuvre une relation de «un vers plusieurs» objets afin que plusieurs objets puissent être notifiés du changement d'état d'un objet et puisse réagir.

Il est très utilisé en IHM, mais peut être appliqué dans bien d'autres cas.

Pattern Observer : Solution



«Pattern Observer» : les rôles

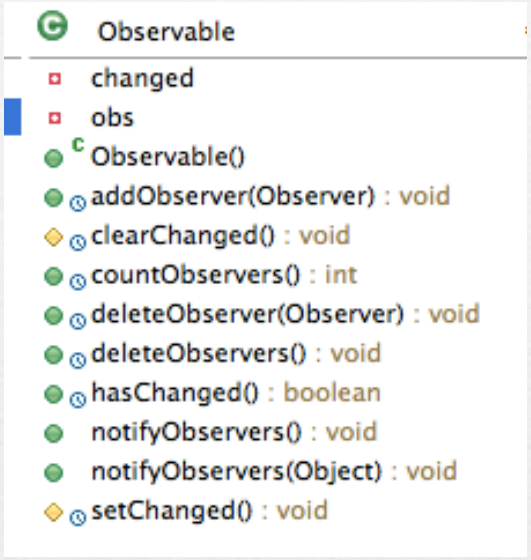
- ✓Rôles : Un sujet et des «observers» (le sujet doit devenir «observable»)
- ✓Responsabilités :
  - ➡Le sujet :
    - notifie les observeurs quand il «change»
    - permet aux observeurs de s'(de-)enregistrer.
  - ➡Les observeurs
    - acceptent les notifications

Pattern Observer : Solution

- **Sujet Abstrait** (Observable) :
  - gère les observeurs (*AddObserver(Observer)*)
  - notifie les observeurs *(notifyObservers)*
- **Sujet (Observable) :**
  - A chaque changement d'état, il «notifie» les observeurs (*notify*)
  - Il peut donner son état (*getState*)
- **Observeur**
  - Se met à jour quand il est notifié (update)
- **Observeur (Abstrait)**
  - Peut être notifié (*update*)

«Pattern Observer» en java

✓Le sujet abstrait : classe abstraite
java.util.observable

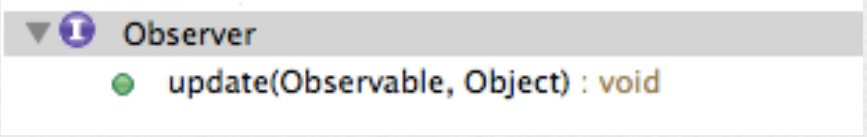


✓Le sujet concret :

➡Votre classe qui hérite de observable

➡*C'est à vous d'appeler* **notifyObservers**

✓L'observeur abstrait : Interface
java.util.Observer



✓L'observeur concret :

➡Votre classe qui «implémente» Observer

➡*doit implémenter* **update**

«Pattern Observer» en java exemple

```
import java.util.Observable;

public class ObservableObject extends Observable
{
    private int n = 0;
    public ObservableObject(int n)
    {
        this.n = n;
    }
    public void setValue(int n)
    {
        this.n = n;
        setChanged();
        notifyObservers();
    }
    public int getValue()
    {
        return n;
    }
}
```

http://www.javaworld.com/article/2077258/learn-java/observer-and-observable.html

«Pattern Observer» en java exemple

```
   import java.util.Observer;
import java.util.Observable;
public class TextObserver implements Observer
{
   private ObservableObject ov = null;
   public TextObserver(ObservableObject ov)
   {
      this.ov = ov;
   }
   public void update(Observable obs, Object obj)
   {
      if (obs == ov)
      {
         System.out.println(ov.getValue());
      }
   }
}
```

http://www.javaworld.com/article/2077258/learn-java/observer-and-observable.html

«Pattern Observer» en java exemple

```
public class Main
{
    public Main()
    {
        ObservableValue ov = new ObservableValue(0);
        TextObserver to = new TextObserver(ov);
        ov.addObserver(to);
    }
    public static void main(String [] args)
    {
        Main m = new Main();
    }
}
```

http://www.javaworld.com/article/2077258/learn-java/observer-and-observable.html

«Pattern Observer» en action

- ✓Un forum
  - ➡On peut poster des messages dans le forum : un message à un titre.
- ✓Des abonnés
  - ➡Un abonné peut recevoir des messages dans ses boites de messages.
- ✓Dès qu'un message est posté sur le forum, tous les abonnés sont notifiés.
  - ➡Certains abonnés enregistrent le message dans leur boite.
  - ➡(2) Certains abonnés n'enregistrent que les messages dont le titre contient «IUT» ;-)

DP Observateur : Résumé

✓**Intention** :

➡ Définit une interdépendance de type un à plusieurs, de telle façon que quand un objet change d'état, tous ceux qui en dépendant en soient notifiés et automatiquement mis à jour.

✓**Applicabilité** : Utilisez l'Observateur dans les situations suivantes :

➡Quand un concept a deux représentations, l'une dépendant de l'autre. Encapsuler ces deux représentations dans des objets distincts permet de les réutiliser et de les modifier indépendamment.

➡Quand la modification d'un objet nécessite de modifier les autres, et que l'on ne sait pas combien sont ces autres.

➡- Quand un objet doit être capable de faire une notification à d'autres objets sans faire d'hypothèses sur la nature de ces objets. En d'autres termes, quand ces objets ne doivent pas être trop fortement couplés.

http://www.goprod.bouhours.net/?page=pattern&pat_id=16

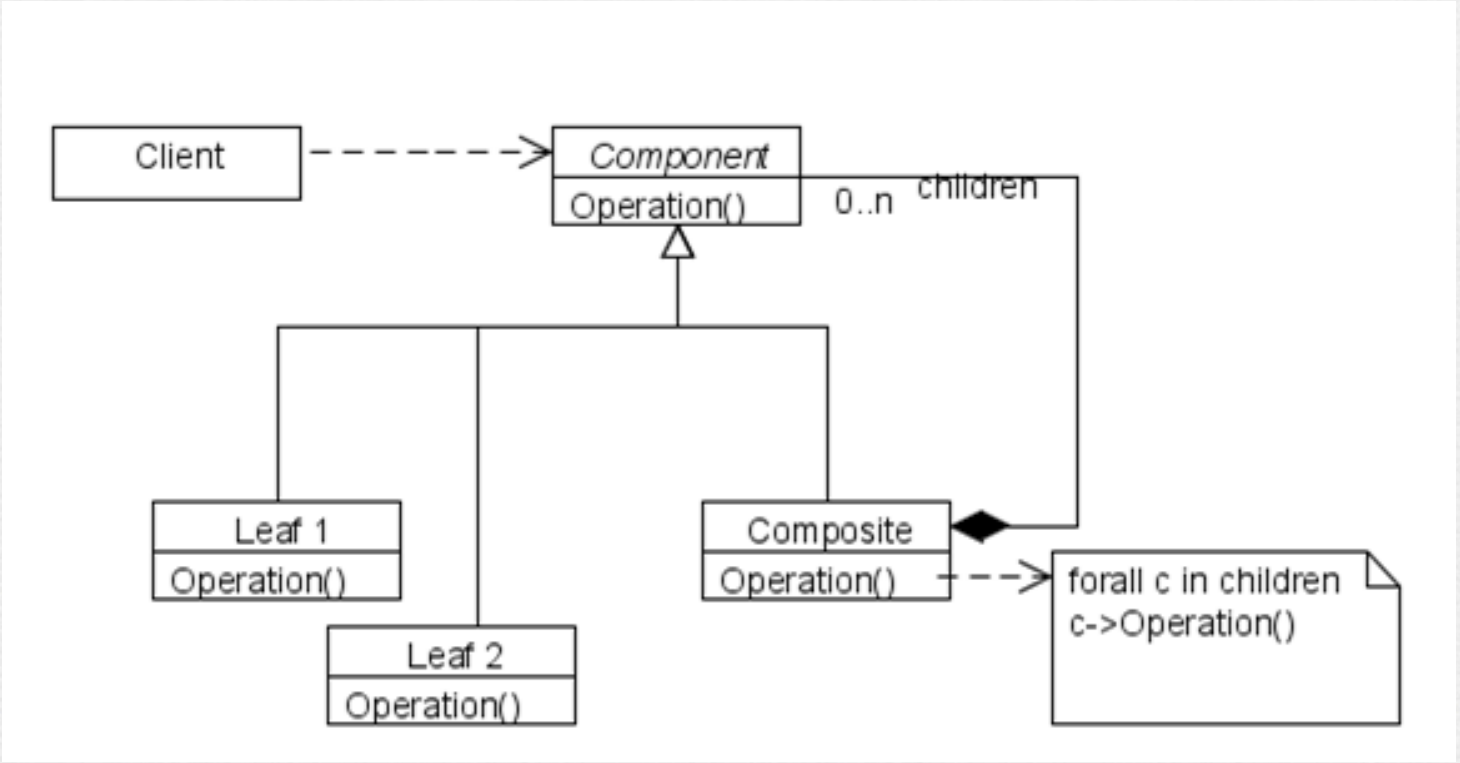
Design Pattern Composite

«Pattern Composite» : le problème

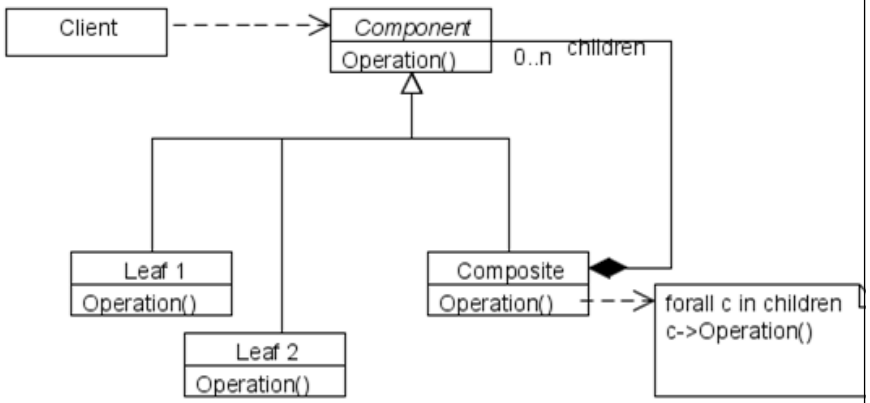
- Problème :
  - Une application doit manipuler des collections d'objets dont certains sont «primitifs» et d'autres «composites».
  - Ces objets doivent tous répondre à une méthode dont le comportement est différent selon que l'objet est composite ou non.

«Pattern Composite» : la solution

- ✓Organiser les objets dans une structure d'arbre qui capture la hiérarchie «partie-contenant».
- ✓Le client adresse alors tous les objets de manière uniforme. On parle de composition récursive.



«Pattern Composite» : les rôles

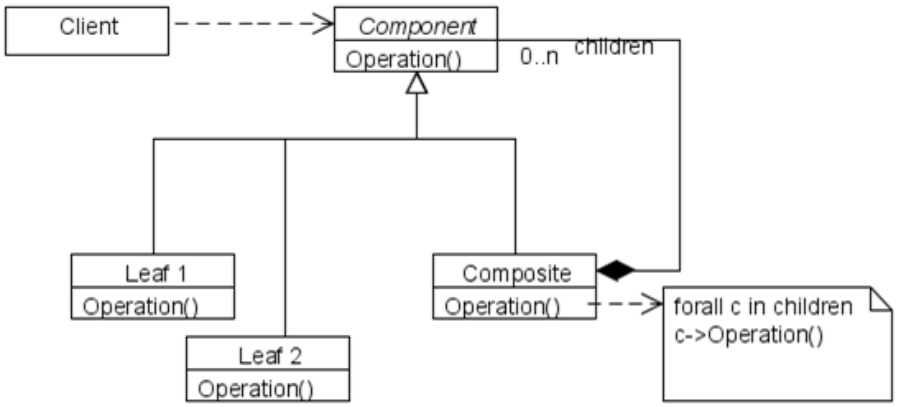


- **Component**
  - declare l'interface des objets pris en compte dans la composition
  - implémente le comportement par défaut des composants autant que possible
- **Composite**
  - définit le comportement des composants ayant des «enfants» dans la hiérarchie de la composition
  - référence les composants fils (ils peuvent eux-même être composites!)
- **Leaf**
  - définit le comportement des objets primitifs dans la composition
- Client
  - manipule les objets de la composition au travers de l'interface Component

Design Pattern Composite en action

- ✓Notre entreprise vend des produits. Chaque produit a un prix, une référence et une description. Nous vendons des jeux videos. A certaines périodes de l'année nous vendons les jeux par lots. Le prix du lot est alors la somme des prix des jeux dans le lot réduite de 10%.
- ✓Nous construisons notre catalogue comme un ensemble de produits que l'on peut imprimer.
- ✓Tout produit créé a une référence qui est automatiquement déterminée à la création du produit.

A vous !

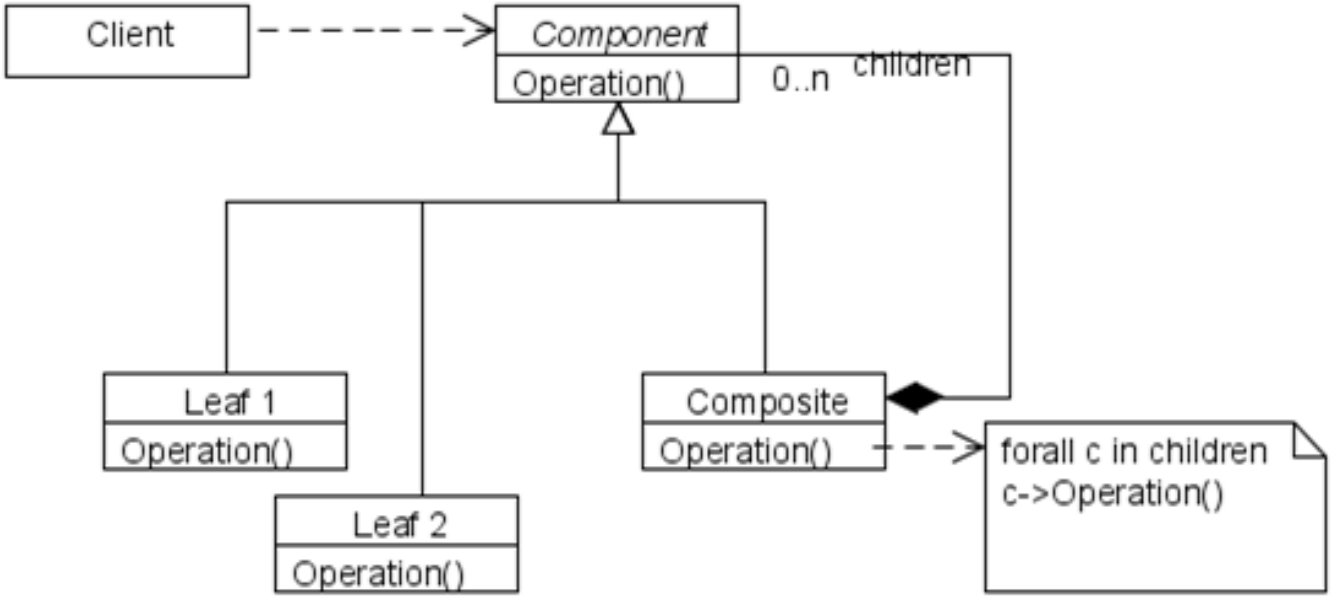


DP Composite : Résumé

- ✓ **Intention :**
  - ➡ Compose des objets en des structures arborescentes pour représenter des hiérarchies composant/composé.
  - ➡ Permet au client de traiter d'une unique façon les objets et les combinaisons d'objets.
- ✓ **Applicabilité** : Utilisez le **Composite** lorsque :
  - ➡ Vous souhaitez représenter des hiérarchies de l'individu.
  - ➡ Vous souhaitez que le client n'ait pas à se préoccuper de la différence entre "combinaisons d'objets" et "objets individuels". Les clients pourront traiter de façon uniforme tous les objets de la structure composite.
- ✓ **Points forts :**
  - ✓ 1. Découplage et extensibilité
  - ✓ 1.1 Factorisation maximale de la composition
  - ✓ 1.2 L'ajout ou la suppression d'une feuille n'implique pas de modification de code
  - ✓ 1.3 L'ajout ou la suppression d'un composite n'implique pas de modification de code
  - ✓ 2. Protocole uniforme
  - ✓ 2.1 Protocole uniforme sur les opérations des objets composés
  - ✓ 2.2 Protocole uniforme sur la gestion de la composition
  - ✓ 2.3 Point d'accès unique pour la classe client

http://www.goprod.bouhours.net/?page=pattern&pat_id=7

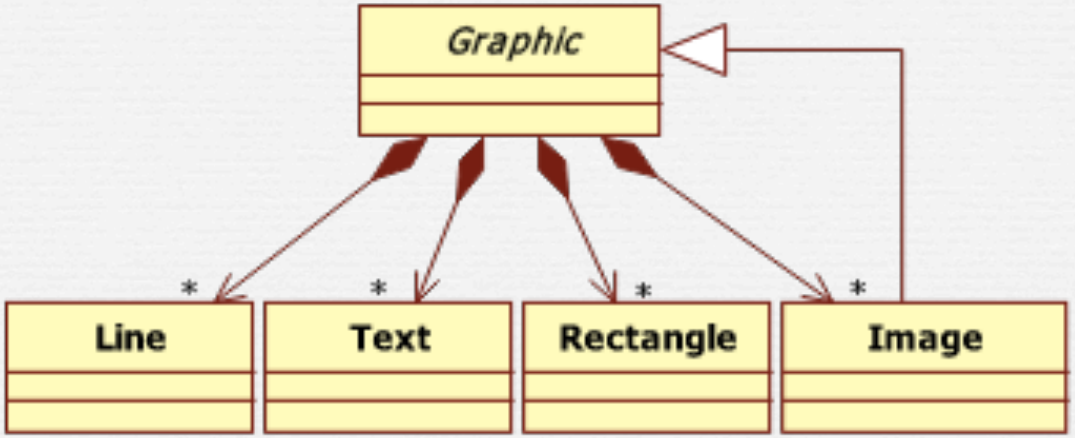
DP Composite : Résumé



✓ **Points forts :**

- ✓ 1. Découplage et extensibilité
- ✓ 1.1 Factorisation maximale de la composition
- ✓ 1.2 L'ajout ou la suppression d'une feuille n'implique pas de modification de code
- ✓ 1.3 L'ajout ou la suppression d'un composite n'implique pas de modification de code
- ✓ 2. Protocole uniforme
- ✓ 2.1 Protocole uniforme sur les opérations des objets composés
- ✓ 2.2 Protocole uniforme sur la gestion de la composition
- ✓ 2.3 Point d'accès unique pour la classe client

http://www.goprod.bouhours.net/?page=pattern&pat_id=7

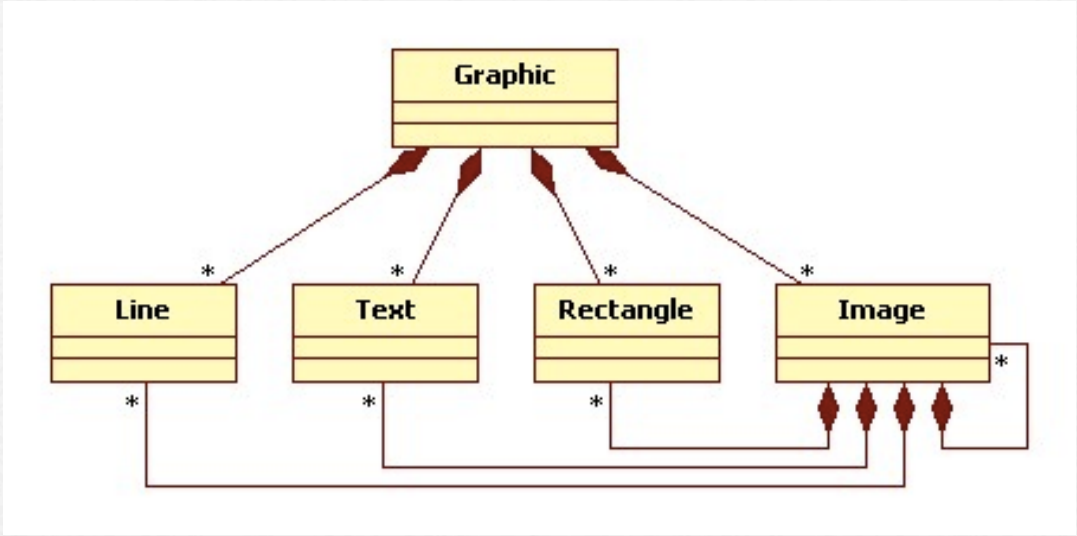


Quels sont les points faibles de ce modèle?

Modéliser un système permettant de dessiner un graphique. Un graphique est composé de lignes, de rectangles, de textes et d'images, une image pouvant être composée d'autres images, de lignes, de rectangles et de textes.

- ☐ 1. Découplage et extensibilité
- ☐ 1.1 Factorisation maximale de la composition
- ☐ 1.2 L'ajout ou la suppression d'une feuille n'implique pas de modification de code
- ☐ 1.3 L'ajout ou la suppression d'un composite ne n'implique pas de modification de code
- ☐ 2. Protocole uniforme
- ☐ 2.1 Protocole uniforme sur les opérations des objets composés
- ☐ 2.2 Protocole uniforme sur la gestion de la composition
- ☐ 2.3 Point d'accès unique pour la classe client

http://www.goprod.bouhours.net/?page=spoiled&sp_id=1

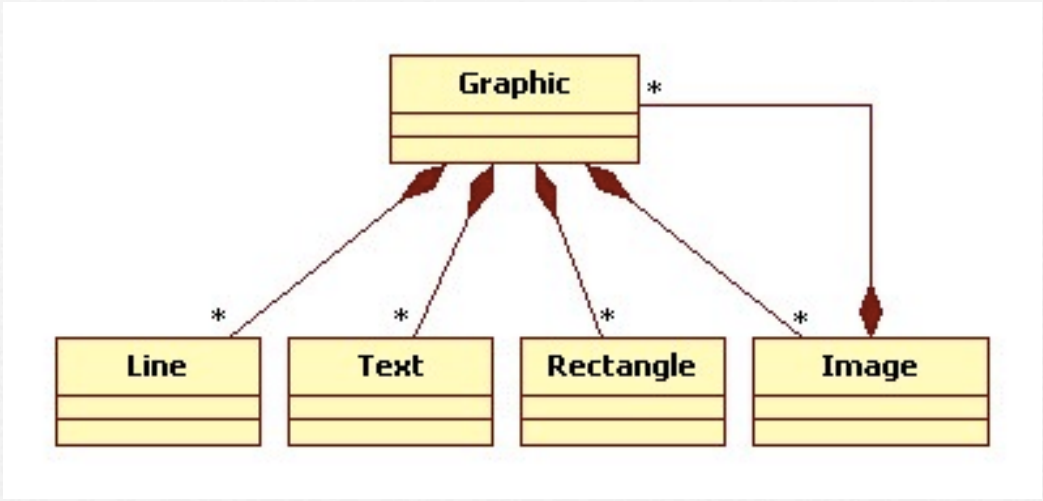


Quels sont les points faibles de ce modèle?

Modéliser un système permettant de dessiner un graphique. Un graphique est composé de lignes, de rectangles, de textes et d'images, une image pouvant être composée d'autres images, de lignes, de rectangles et de textes.

- ☐ 1. Découplage et extensibilité
- ☐ 1.1 Factorisation maximale de la composition
- ☐ 1.2 L'ajout ou la suppression d'une feuille n'implique pas de modification de code
- ☐ 1.3 L'ajout ou la suppression d'un composite ne n'implique pas de modification de code
- ☐ 2. Protocole uniforme
- ☐ 2.1 Protocole uniforme sur les opérations des objets composés
- ☐ 2.2 Protocole uniforme sur la gestion de la composition
- ☐ 2.3 Point d'accès unique pour la classe client

http://www.goprod.bouhours.net/?page=spoiled&sp_id=1

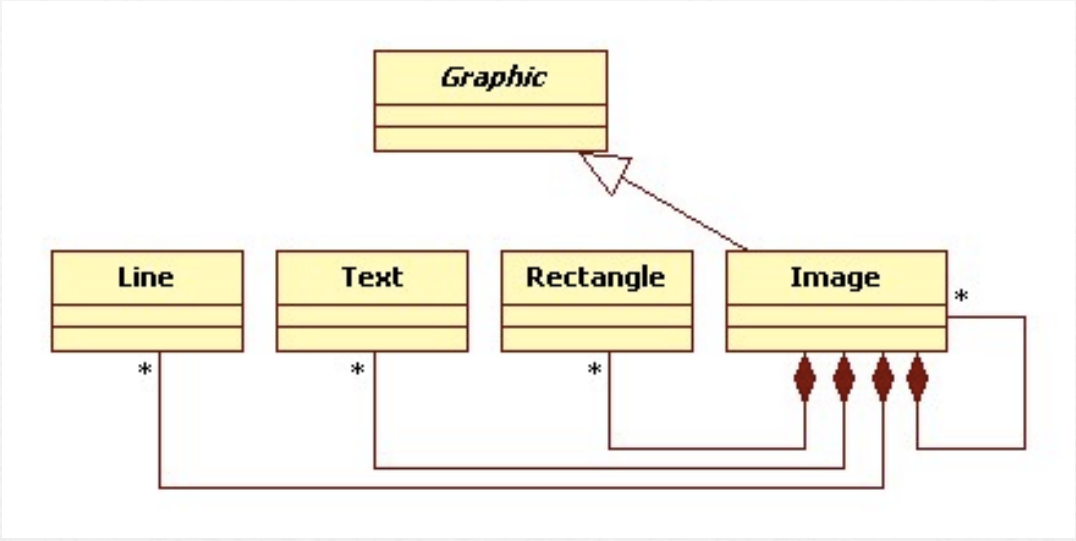


Quels sont les points faibles de ce modèle?

Modéliser un système permettant de dessiner un graphique. Un graphique est composé de lignes, de rectangles, de textes et d'images, une image pouvant être composée d'autres images, de lignes, de rectangles et de textes.

- ☐ 1. Découplage et extensibilité
- ☐ 1.1 Factorisation maximale de la composition
- ☐ 1.2 L'ajout ou la suppression d'une feuille n'implique pas de modification de code
- ☐ 1.3 L'ajout ou la suppression d'un composite ne n'implique pas de modification de code
- ☐ 2. Protocole uniforme
- ☐ 2.1 Protocole uniforme sur les opérations des objets composés
- ☐ 2.2 Protocole uniforme sur la gestion de la composition
- ☐ 2.3 Point d'accès unique pour la classe client

http://www.goprod.bouhours.net/?page=spoiled&sp_id=1

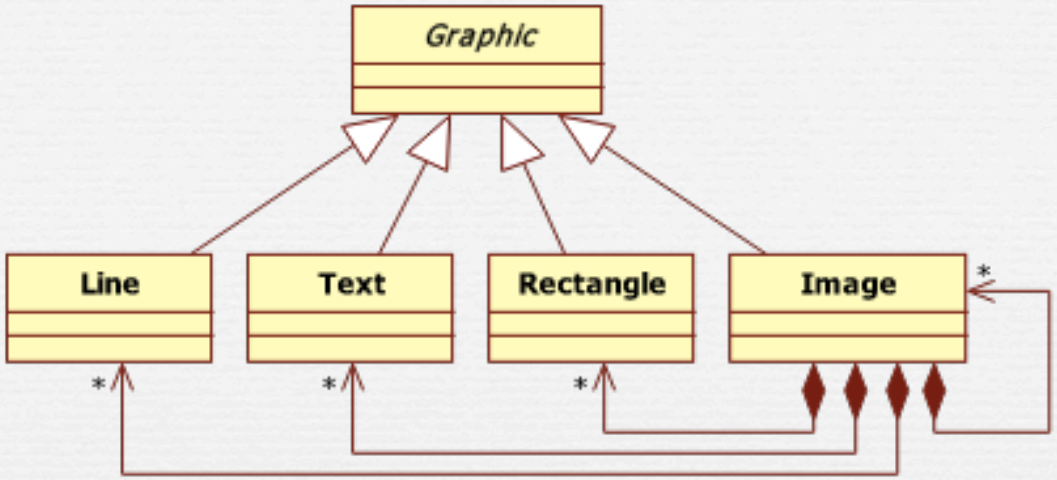


Quels sont les points faibles de ce modèle?

Modéliser un système permettant de dessiner un graphique. Un graphique est composé de lignes, de rectangles, de textes et d'images, une image pouvant être composée d'autres images, de lignes, de rectangles et de textes.

- ☐ 1. Découplage et extensibilité
- ☐ 1.1 Factorisation maximale de la composition
- ☐ 1.2 L'ajout ou la suppression d'une feuille n'implique pas de modification de code
- ☐ 1.3 L'ajout ou la suppression d'un composite ne n'implique pas de modification de code
- ☐ 2. Protocole uniforme
- ☐ 2.1 Protocole uniforme sur les opérations des objets composés
- ☐ 2.2 Protocole uniforme sur la gestion de la composition
- ☐ 2.3 Point d'accès unique pour la classe client

http://www.goprod.bouhours.net/?page=spoiled&sp_id=1

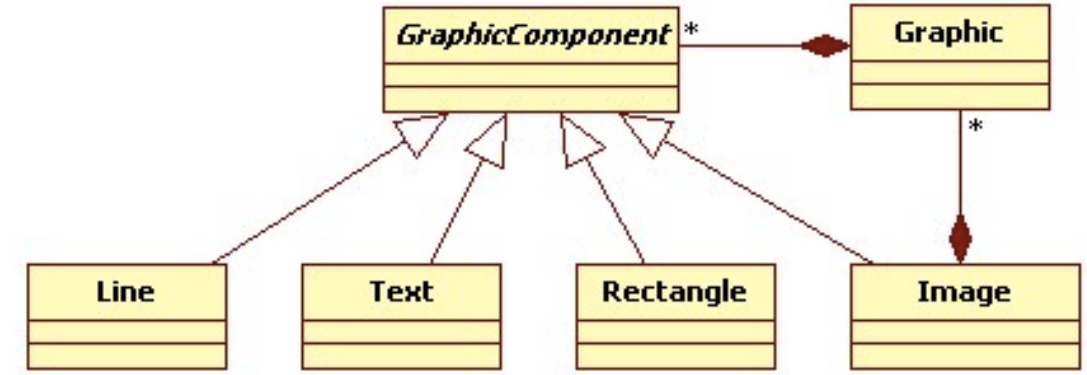


Quels sont les points faibles de ce modèle?

Modéliser un système permettant de dessiner un graphique. Un graphique est composé de lignes, de rectangles, de textes et d'images, une image pouvant être composée d'autres images, de lignes, de rectangles et de textes.

- [ ] 1. Découplage et extensibilité
- [ ] 1.1 Factorisation maximale de la composition
- [ ] 1.2 L'ajout ou la suppression d'une feuille n'implique pas de modification de code
- [ ] 1.3 L'ajout ou la suppression d'un composite ne n'implique pas de modification de code
- [ ] 2. Protocole uniforme
- [ ] 2.1 Protocole uniforme sur les opérations des objets composés
- [ ] 2.2 Protocole uniforme sur la gestion de la composition
- [ ] 2.3 Point d'accès unique pour la classe client

http://www.goprod.bouhours.net/?page=spoiled&sp_id=1



Quels sont les points faibles de ce modèle?

Modéliser un système permettant de dessiner un graphique. Un graphique est composé de lignes, de rectangles, de textes et d'images, une image pouvant être composée d'autres images, de lignes, de rectangles et de textes.

- [ ] 1. Découplage et extensibilité
- [ ] 1.1 Factorisation maximale de la composition
- [ ] 1.2 L'ajout ou la suppression d'une feuille n'implique pas de modification de code
- [ ] 1.3 L'ajout ou la suppression d'un composite ne n'implique pas de modification de code
- [ ] 2. Protocole uniforme
- [ ] 2.1 Protocole uniforme sur les opérations des objets composés
- [ ] 2.2 Protocole uniforme sur la gestion de la composition
- [ ] 2.3 Point d'accès unique pour la classe client

http://www.goprod.bouhours.net/?page=spoiled&sp_id=1

Design Pattern Adaptator

Patron Adapter : le problème

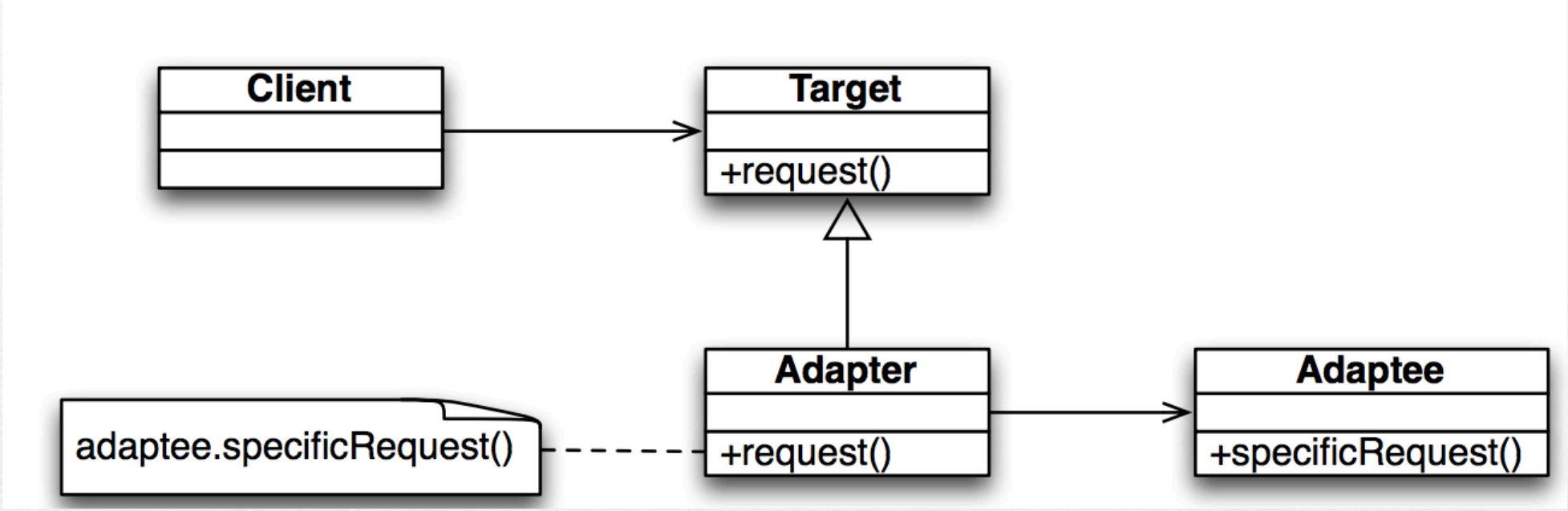
- “ Permettre l’adaptation d’une classe à une autre interface qui est attendue par le client : autoriser ainsi des classes ayant des interfaces incompatibles à collaborer. ”
- Synonyme : wrapper

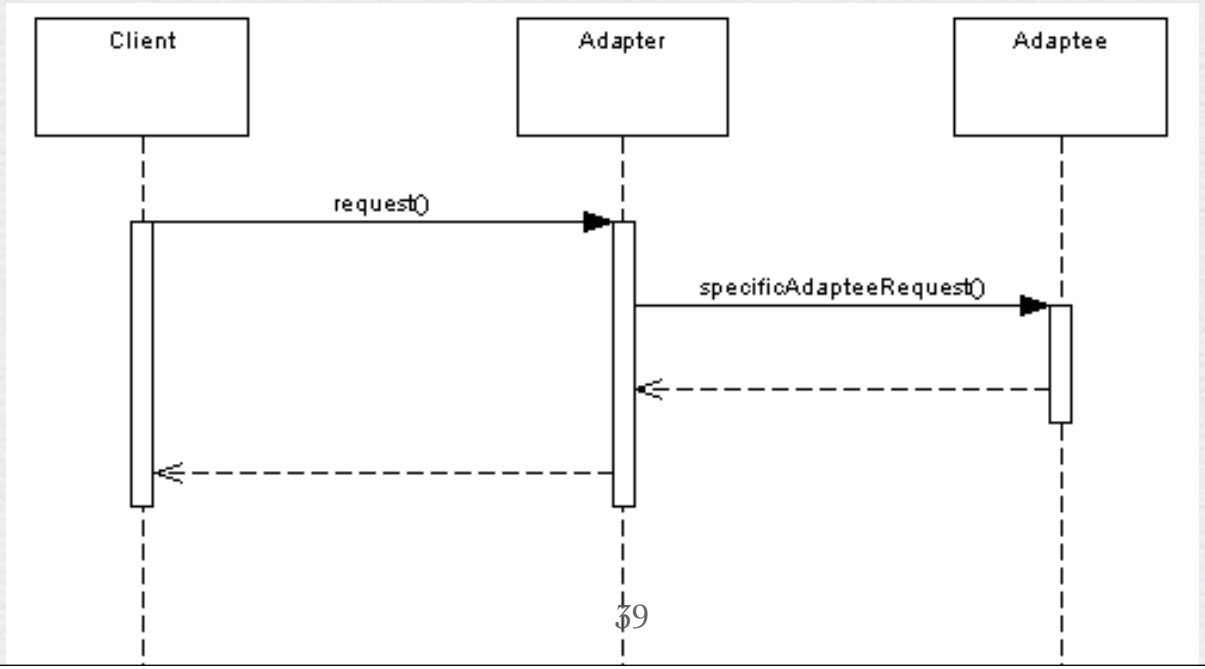
motivation

Parfois, une bibliothèque de code ne peut pas être utilisée car son interface est incompatible avec l'interface requise par une application. Nous ne pouvons pas changer l'interface de la bibliothèque, puisque nous ne pouvons pas accéder ou modifier le code source .... De même il ne serait pas raisonnable d’avoir autant de versions d’une bibliothèque que de ses usages.

http://miageprojet2.unice.fr/User:SimonUrli/Patrons_de_Conception

Patron Adapter : solution





- On a défini un jeu dans lequel des avatars peut avancer, être affecté à un joueur, ou se battre.
- On voudrait pouvoir utiliser des objets présents dans un autre jeu.
- On a déjà les codes suivants :

```
public class Robot {

   private String maitre;

   public void frapper() {..}

   public void marcher() {...}

   public void reagirAHumain(String maitre) {
      this.maitre = maitre; ..
   }
}
```

```
public interface Avatar {
   //Renvoie un nombre de Points
   int attaquer();
   void avancer();
   void affecterJoueur();
}
```

```
public class Tank {

   //Force du tir
   public int tirer() {
   }

   public void rouler(int
vitesse, Direction d) {

   }

   public void
setConducteur(String maitre) {
   }
}
```

Inspiré de : https://www.youtube.com/watch?v=qG286LQM6BU