ADAPTER

Design Pattern

1. Introduction aux patterns
2. Modélisation d’un problème
3. Description du pattern
4. Exemple de code

Qu’est ce qu’un Design Pattern ?

« C’est un modèle spécifique représentant d'une façon schématique la structure d'un comportement individuel. »

1977 A Pattern Language

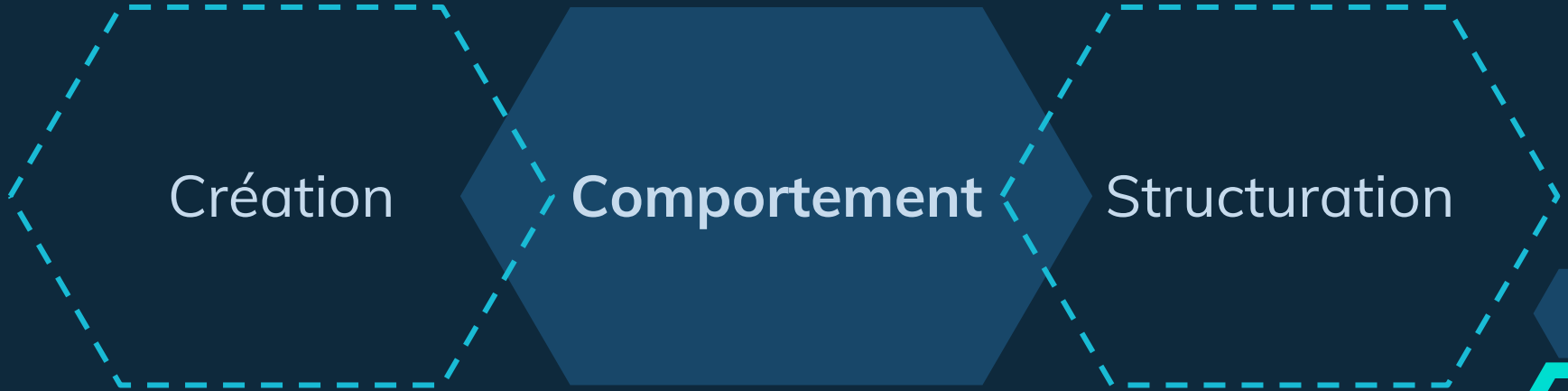
Christopher Alexander, Sara Ishikawa et Murray Silverstein

1987 Kent Beck et Ward Cunningham

1994 GoF

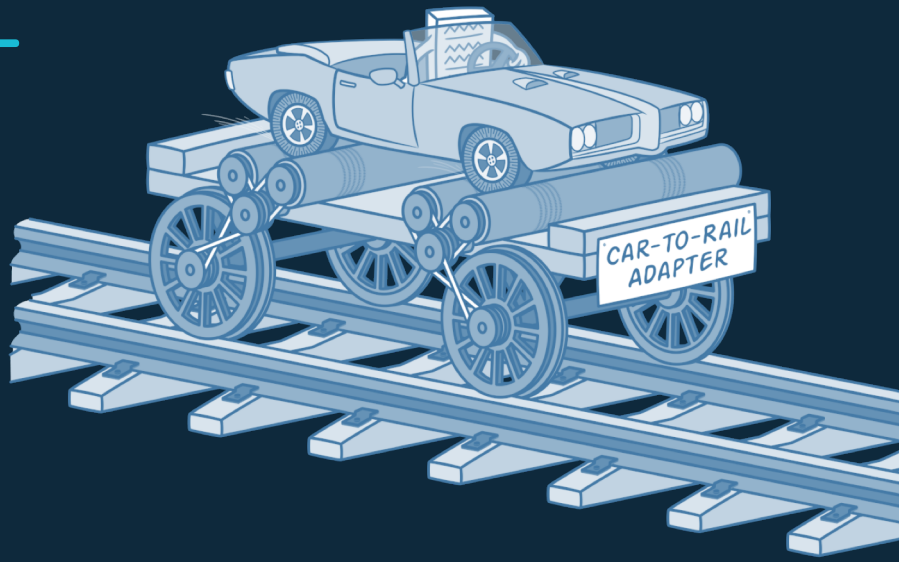
Erich Gamma, Richard Helm, Ralph Johnson et John Vlissides.

Classification des Design Pattern

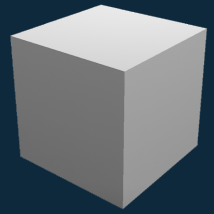


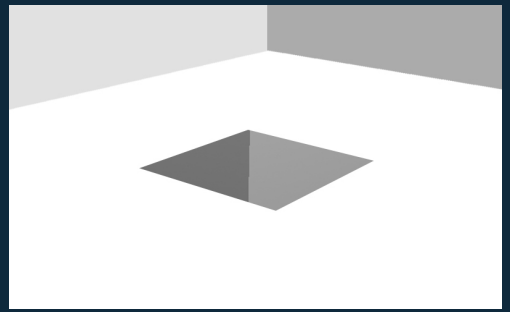
Comment faire interagir des objets incompatibles entre eux ?

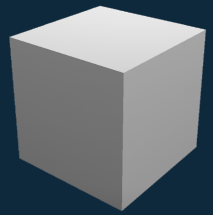
Pattern Adapter

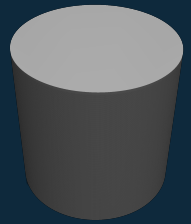


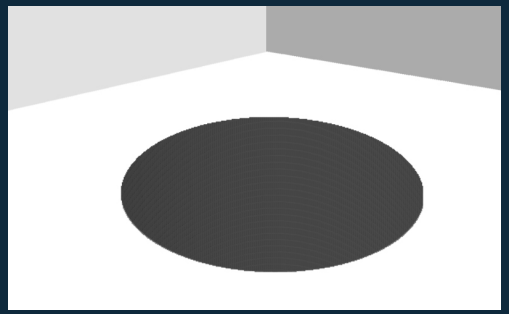
Principe

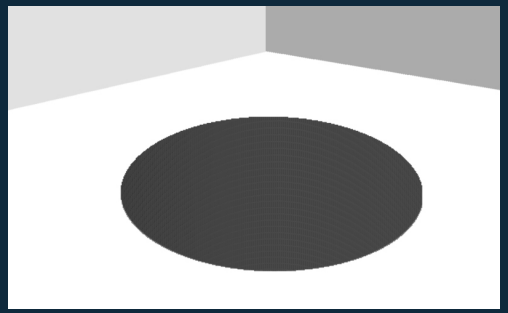




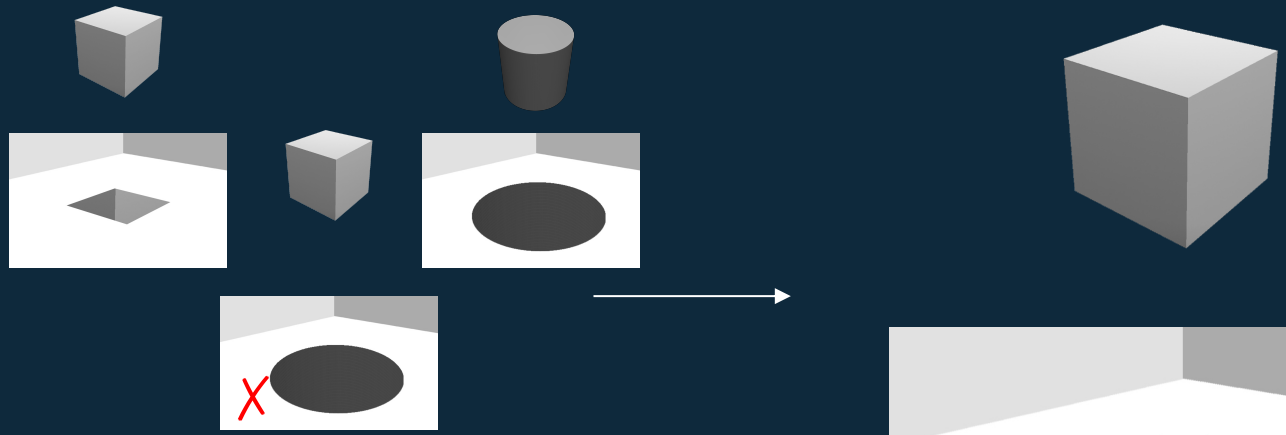








Principe



Modélisation

Dessiner une ligne

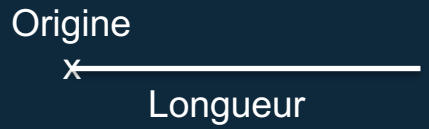
Point A
x

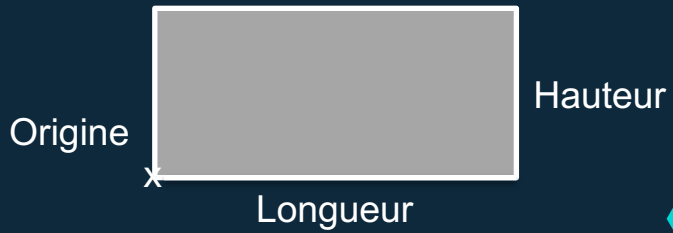
Point B
x

*Un point à 2 coordonnées abscisse et ordonnée dans un plan

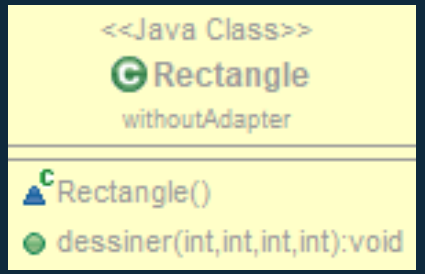
Dessiner un rectangle

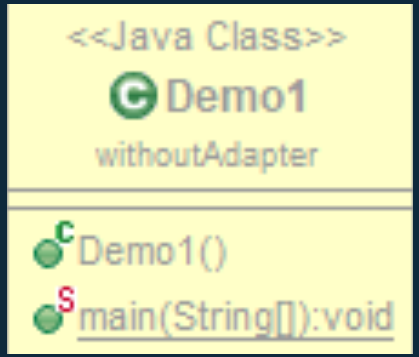
Origine
x





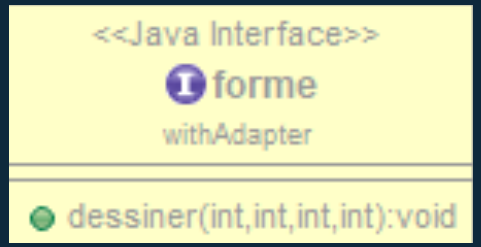
Modélisation



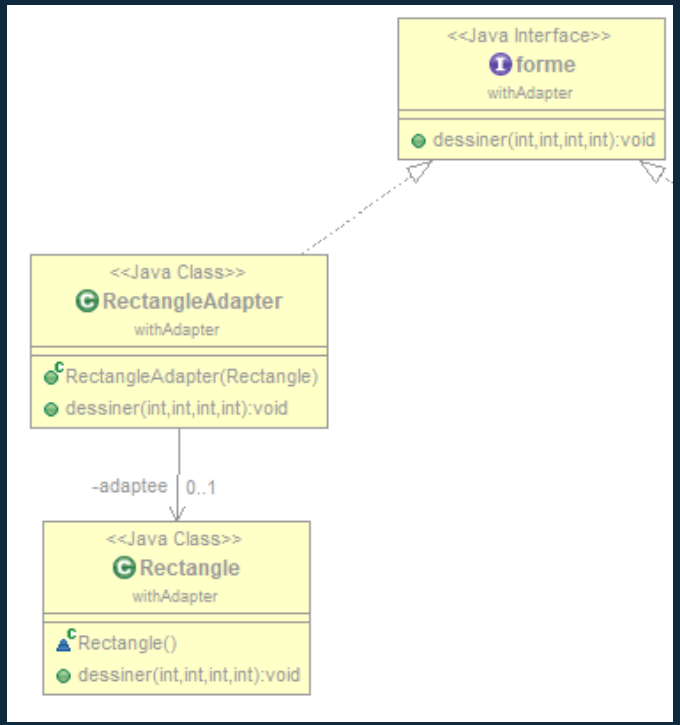


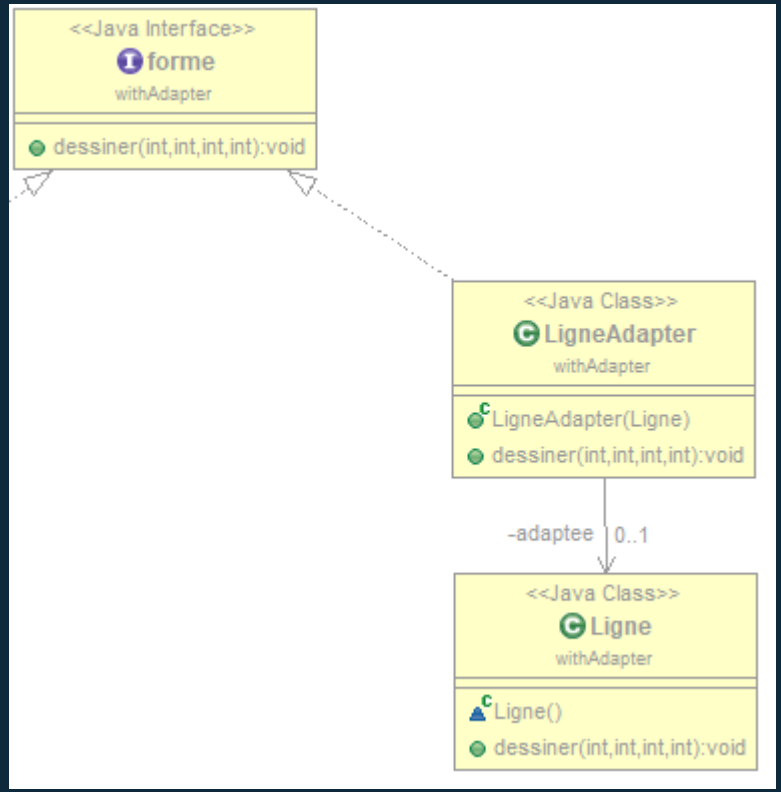
~~OCP~~

Modélisation alternative

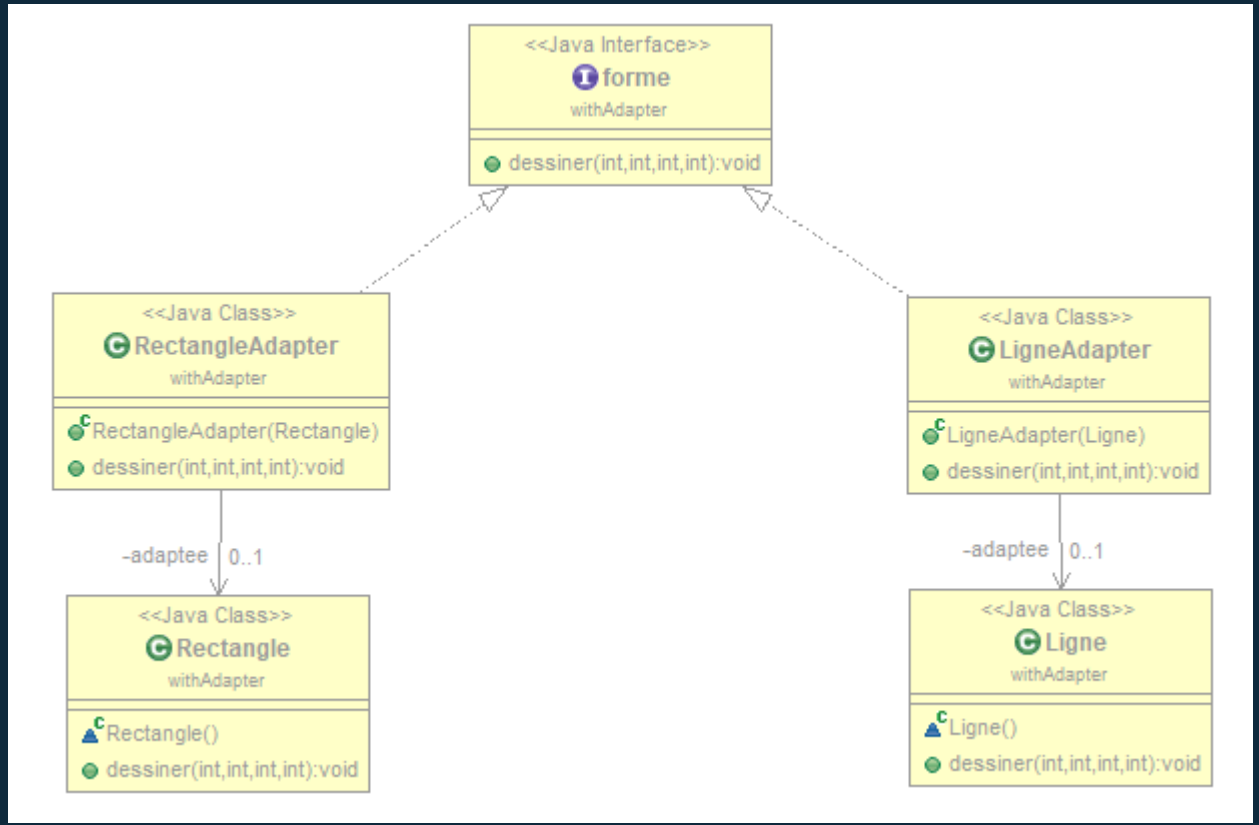


Modélisation alternative



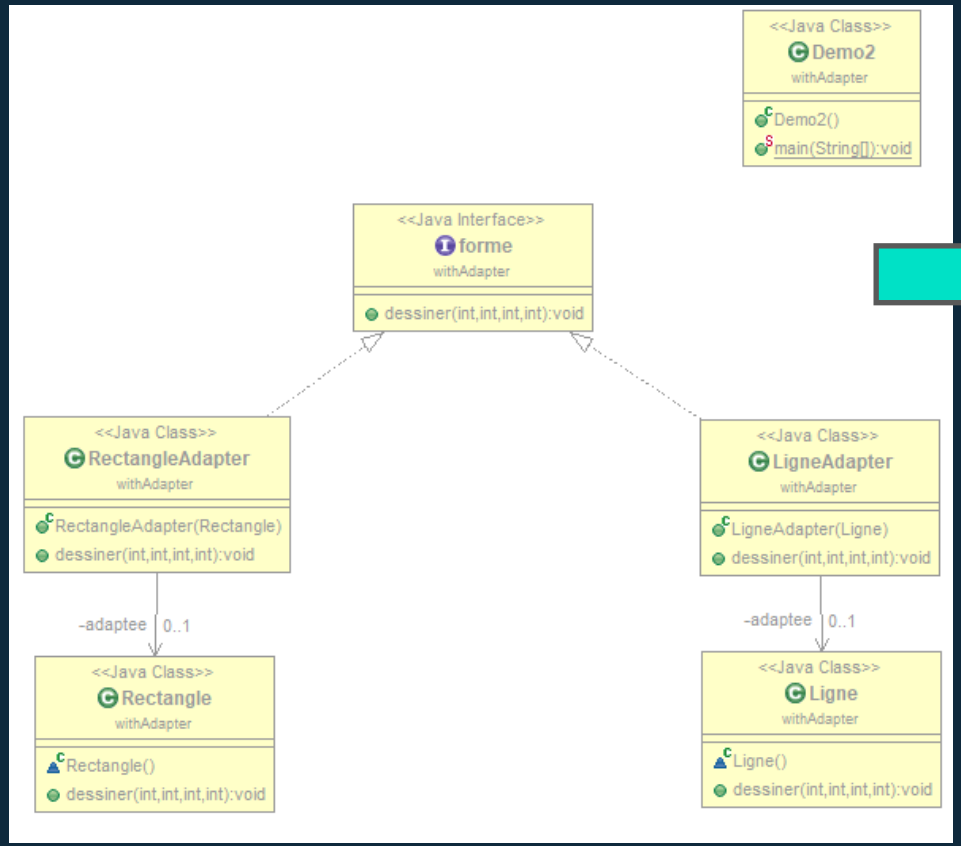


Modélisation alternative



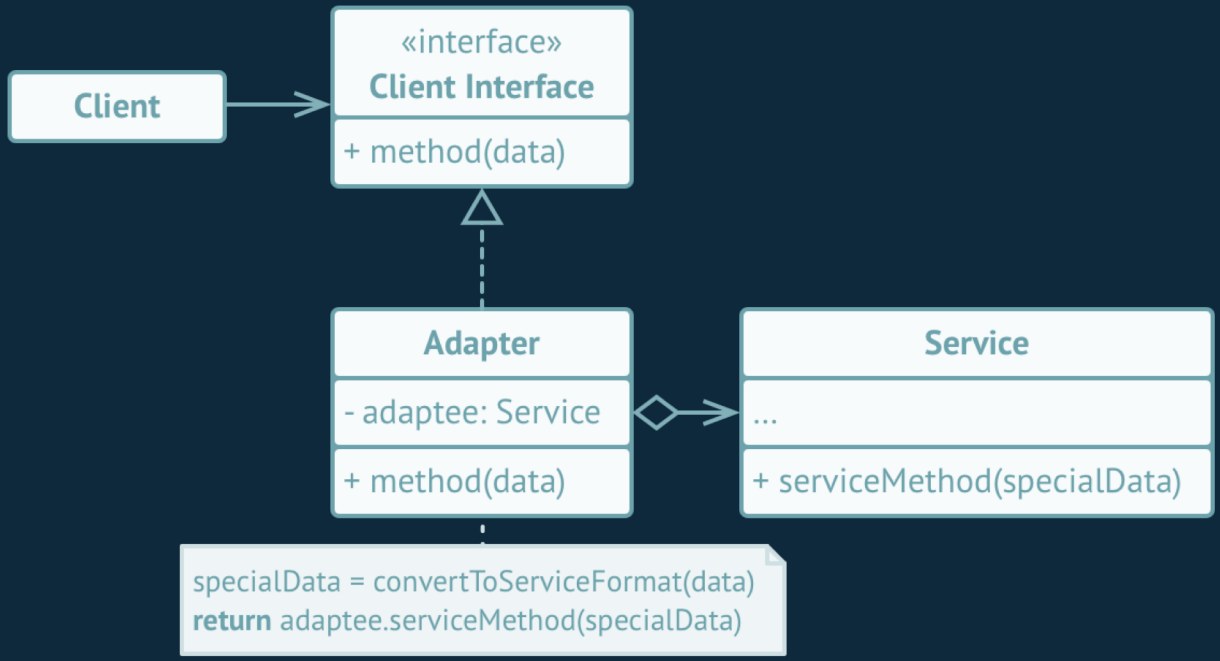
Modélisation alternative

OCP

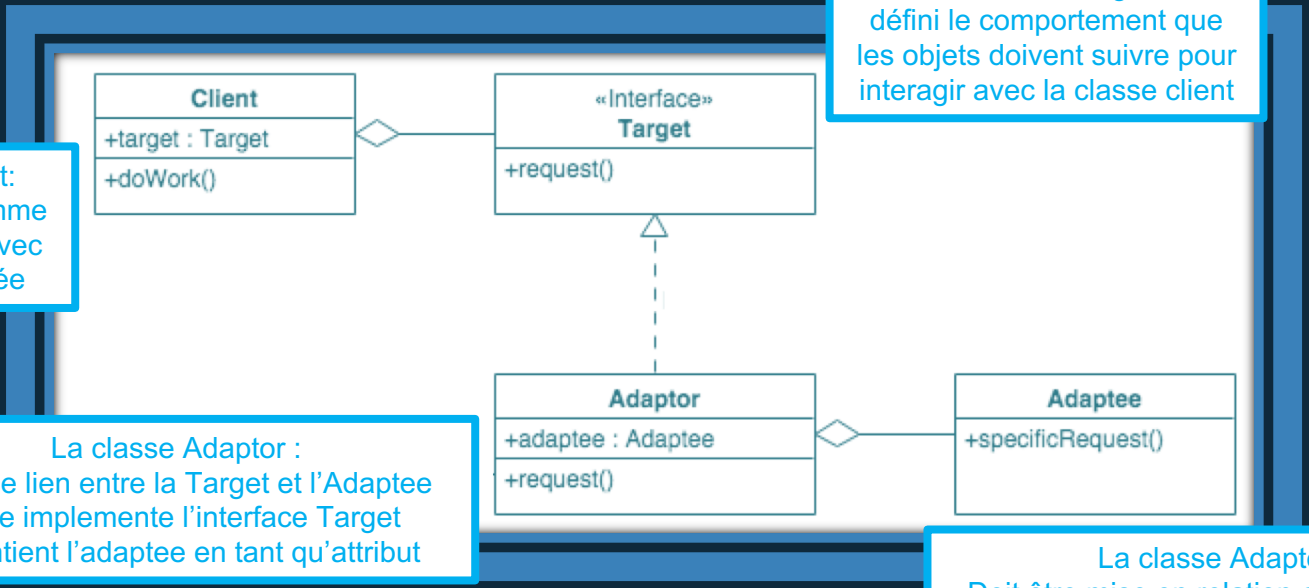


Pattern Adapter

Diagramme de classe



Comment fonctionne le pattern adapter ?



L'interface Target :
défini le comportement que les objets doivent suivre pour interagir avec la classe client

La classe client:
partie du programme qui va interagir avec la classe adaptée

La classe Adaptor :
- Fait le lien entre la Target et l'Adaptee
- elle implemente l'interface Target
- Contient l'adaptee en tant qu'attribut

La classe Adaptee :
Doit être mise en relation avec la classe Client sans implémenter l'interface Target

Single Responsability Principle

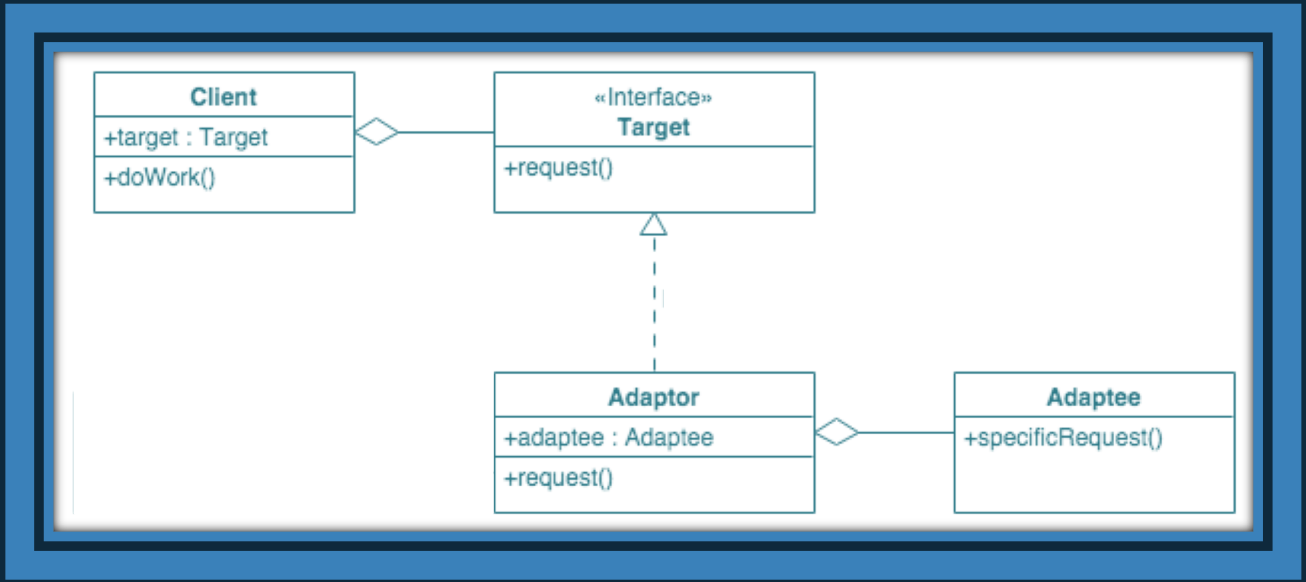
Open Closed Principle

Liskov Substitution Principle

Interface Segregation Principle

Dependancy Inversion Principle

Les principes SOLID



SRP ✓

OCP ✓

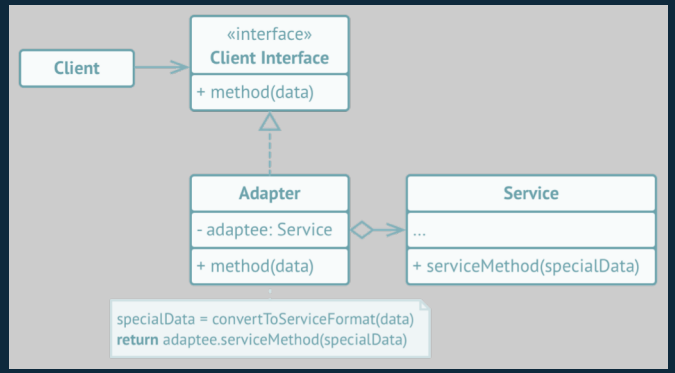
ISP ✓

Avantages et inconvénients

Avantages	Inconvénients
Permet de réduire le nombre de classes vides	Augmente fortement la complexité du programme si trop utilisé
Empêcher la redondance des méthodes	Impossible d'avoir plusieurs adapteurs sur une même classe
Améliorer la lisibilité et la logique métier du code	

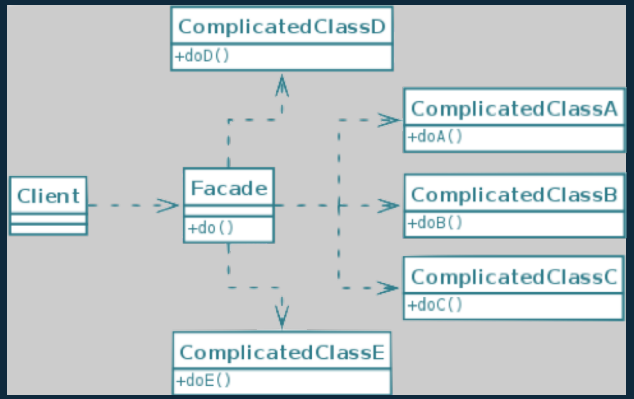
Comparaison

Pattern Adapter

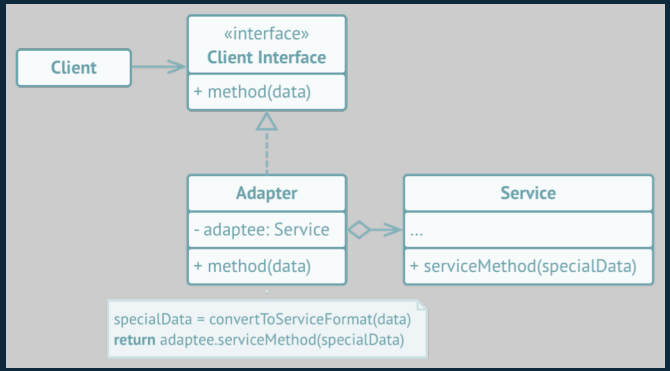


VS

Pattern Façade

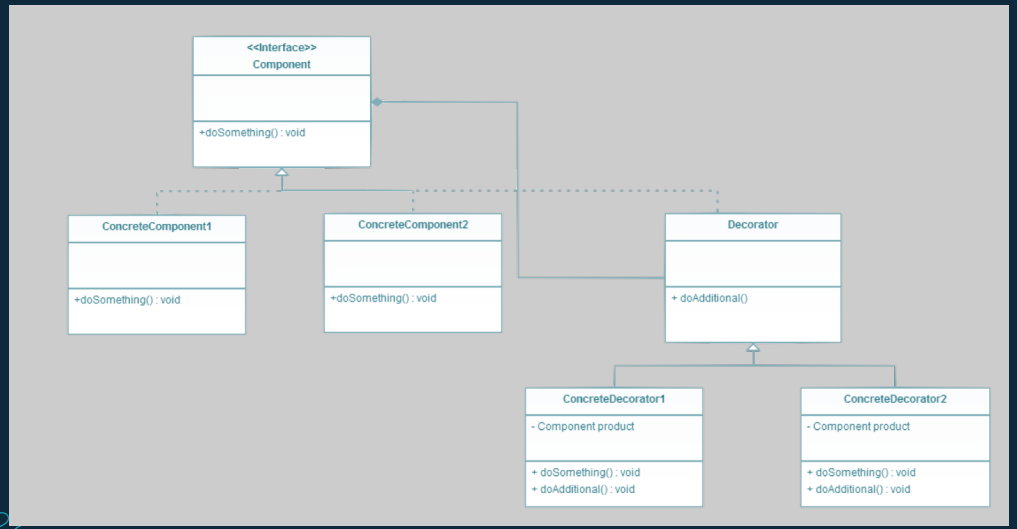


Pattern Adapter

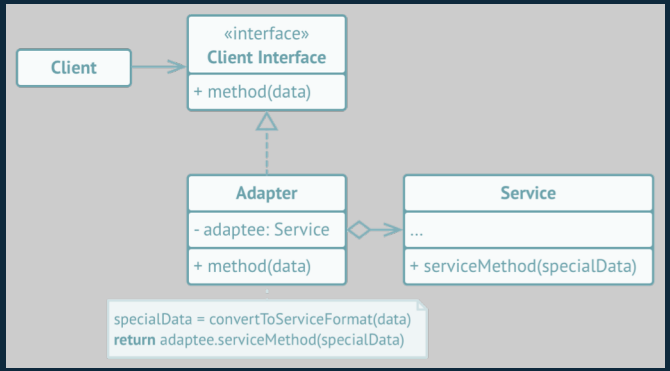


Pattern Decorator

VS

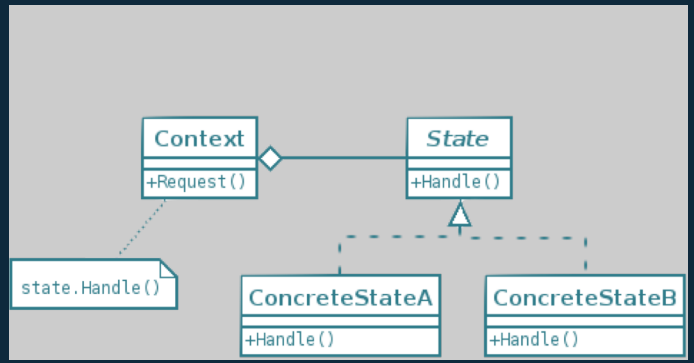


Pattern Adapter

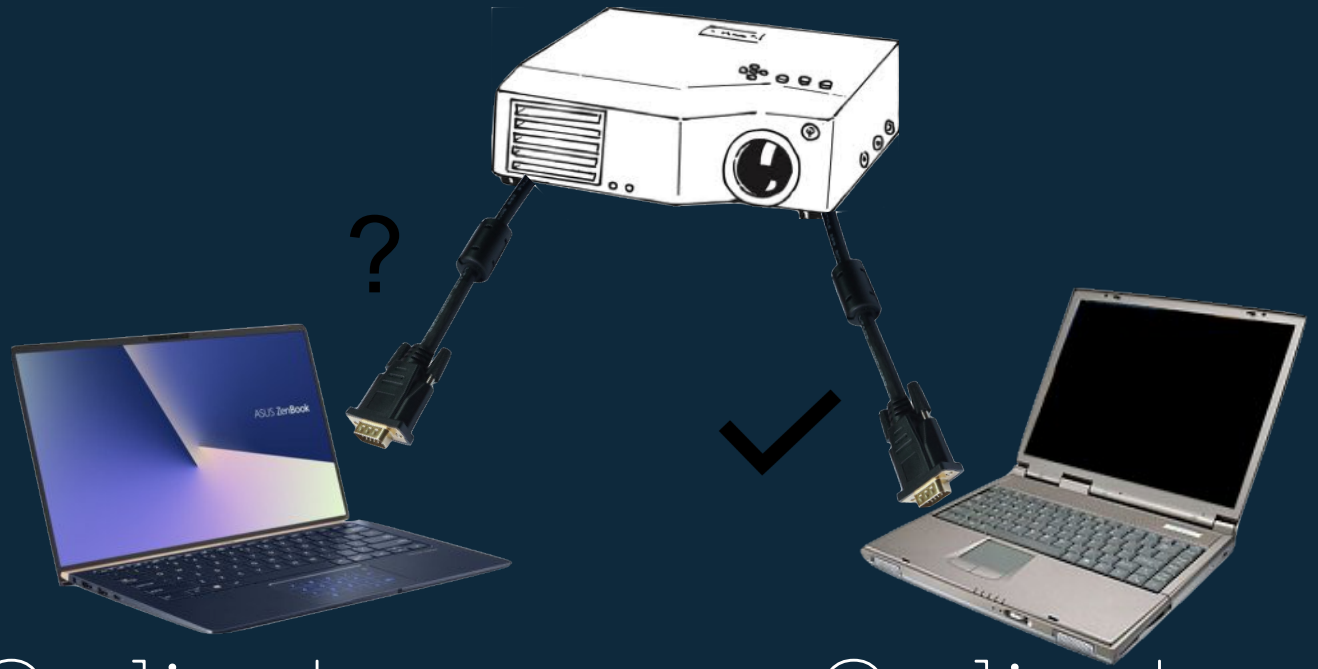


VS

Pattern State



Exemple



Ordinateur HDMI

Ordinateur VGA

Le problème

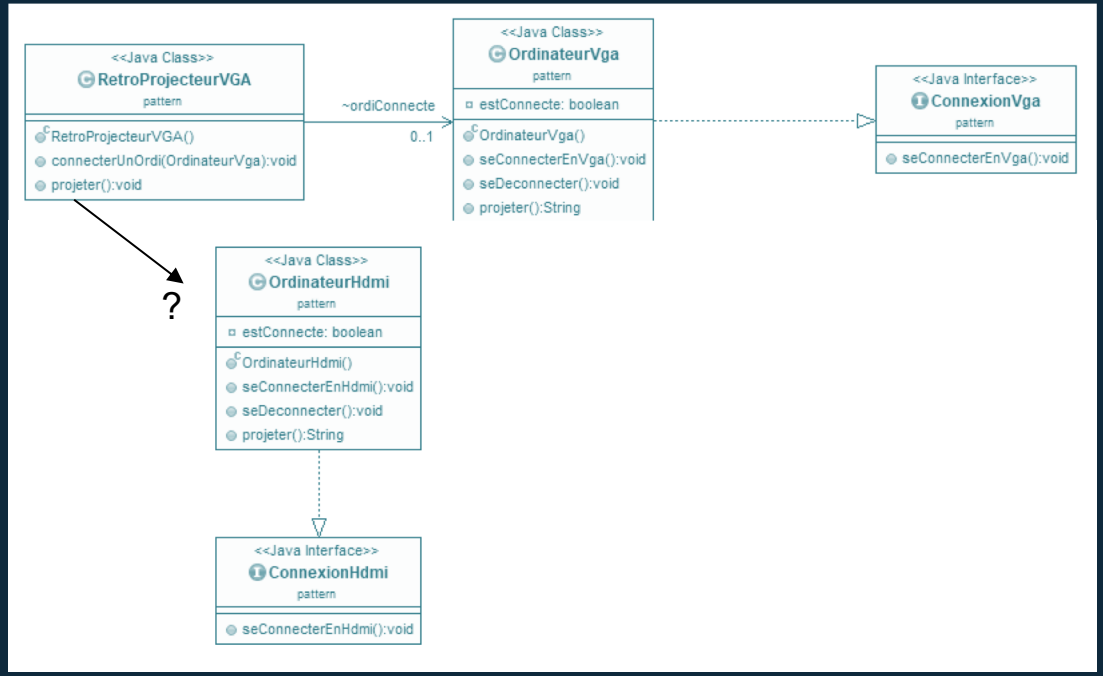
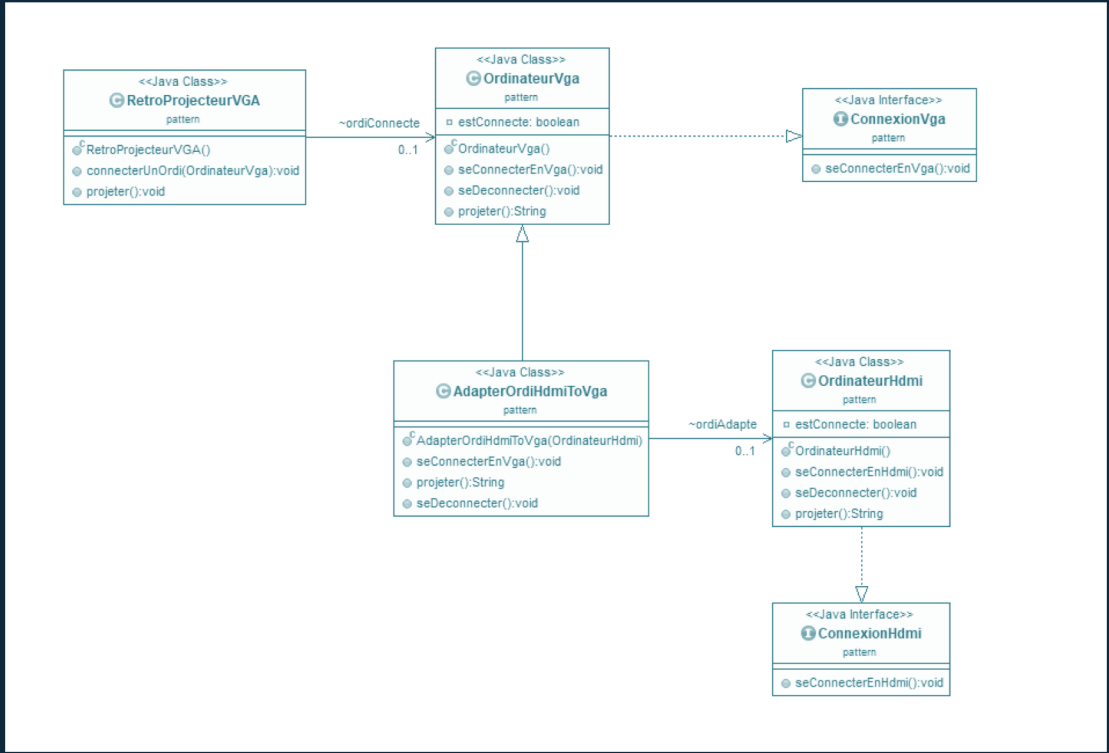

<<Java Class>>
RetroProjecteurVGA
pattern
RetroProjecteurVGA()
connecterUnOrdi(OrdinateurVga):void
projeter():void
~ordiConnecte
0..1
<<Java Class>>
OrdinateurVga
pattern
estConnecte: boolean
OrdinateurVga()
seConnecterEnVga():void
seDeconnecter():void
projeter():String
<<Java Interface>>
ConnexionVga
pattern
seConnecterEnVga():void
?
<<Java Class>>
OrdinateurHdmi
pattern
estConnecte: boolean
OrdinateurHdmi()
seConnecterEnHdmi():void
seDeconnecter():void
projeter():String
<<Java Interface>>
ConnexionHdmi
pattern
seConnecterEnHdmi():void

Diagramme de Classe



Regardons le code maintenant !

Sitographie

Livre :

Les Design Patterns en Java de Steven John Metsker et William C. Wake

Site Web :

https://refactoring.guru/design-patterns/adapter
https://sourcemaking.com/design_patterns/adapter

Vidéo :

Vidéo de Christopher Okhravi : **Adapter Pattern – Design Patterns** :
https://www.youtube.com/watch?v=2PKQtcJjYvc

Vidéo de Derek Banas : **Adapter Pattern Design** :
https://www.youtube.com/watch?v=qG286LQM6BU

QCM

shorturl.at/qG126

Quels principes SOLID le pattern ADAPTER respecte ?

Lequel de ces éléments est une interface ?

A quelle classification appartient le pattern Adapter ?

En quoi peut-on rapprocher le pattern Façade avec le pattern Adapter ?

Quel est l'inconvénient majeur du pattern Adapter ?

Bonus :
Où peut-on acheter un Adaptateur HDMI/VGA SOLID selon Monsieur Chastagner ?

Merci pour votre attention !