PATTERNS DE DÉLÉGATION

Noël Plouzeau

Usages de la délégation

- Extension d'opération
- Extension de méthode
- Joue un rôle dans la réification

Délégation de 1ère classe dans le Gof

- Strategy
- Proxy
- Adapter
- Decorator

LE PATRON STRATEGY

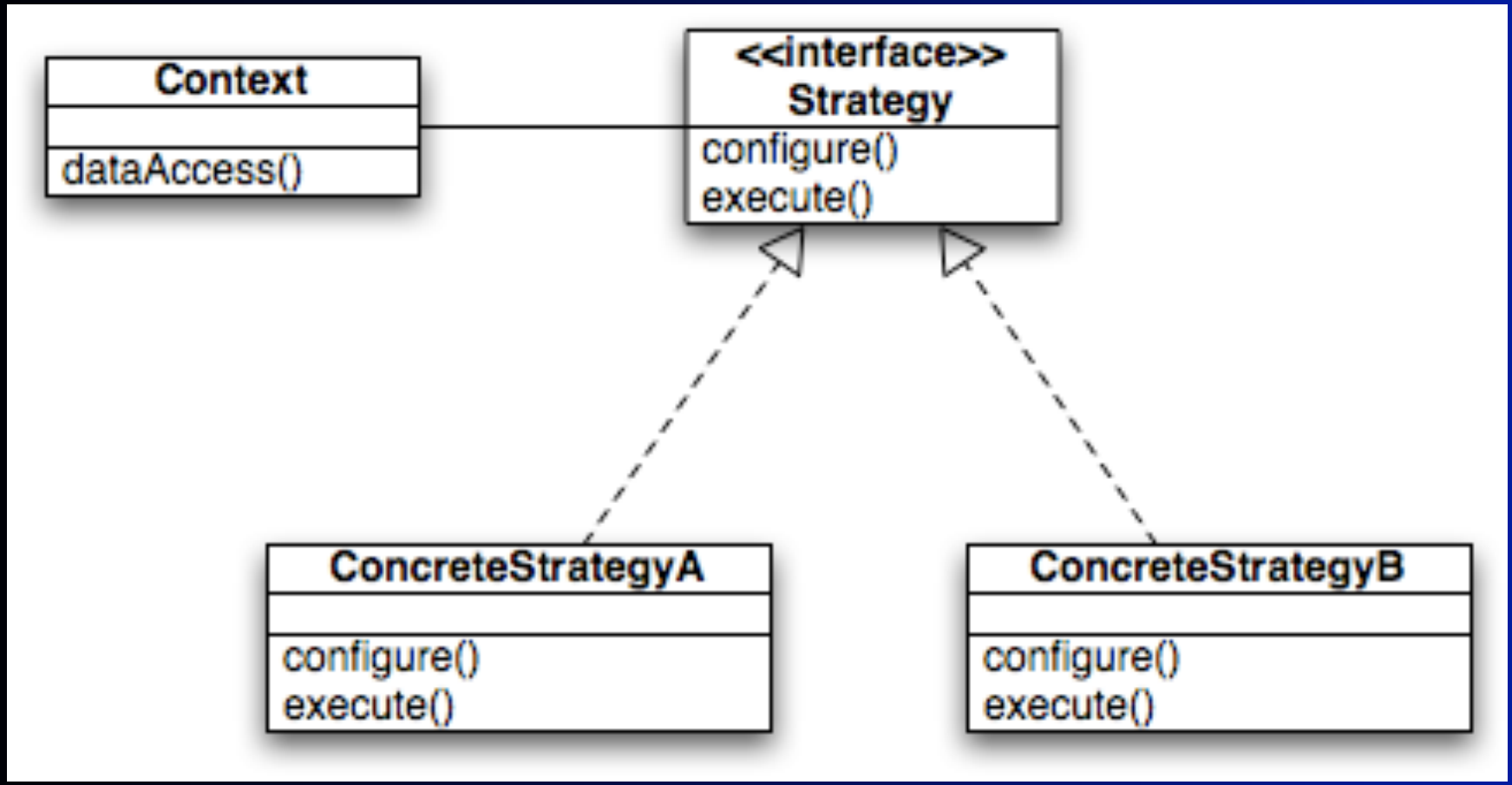
Patron de conception Strategy

- But
  - permettre de mettre en œuvre des algorithmes différents avec un choix dynamique de la mise en œuvre
- Analogie
  - permet de remplacer les « pointeurs de fonction »
- Catégorie
  - comportement

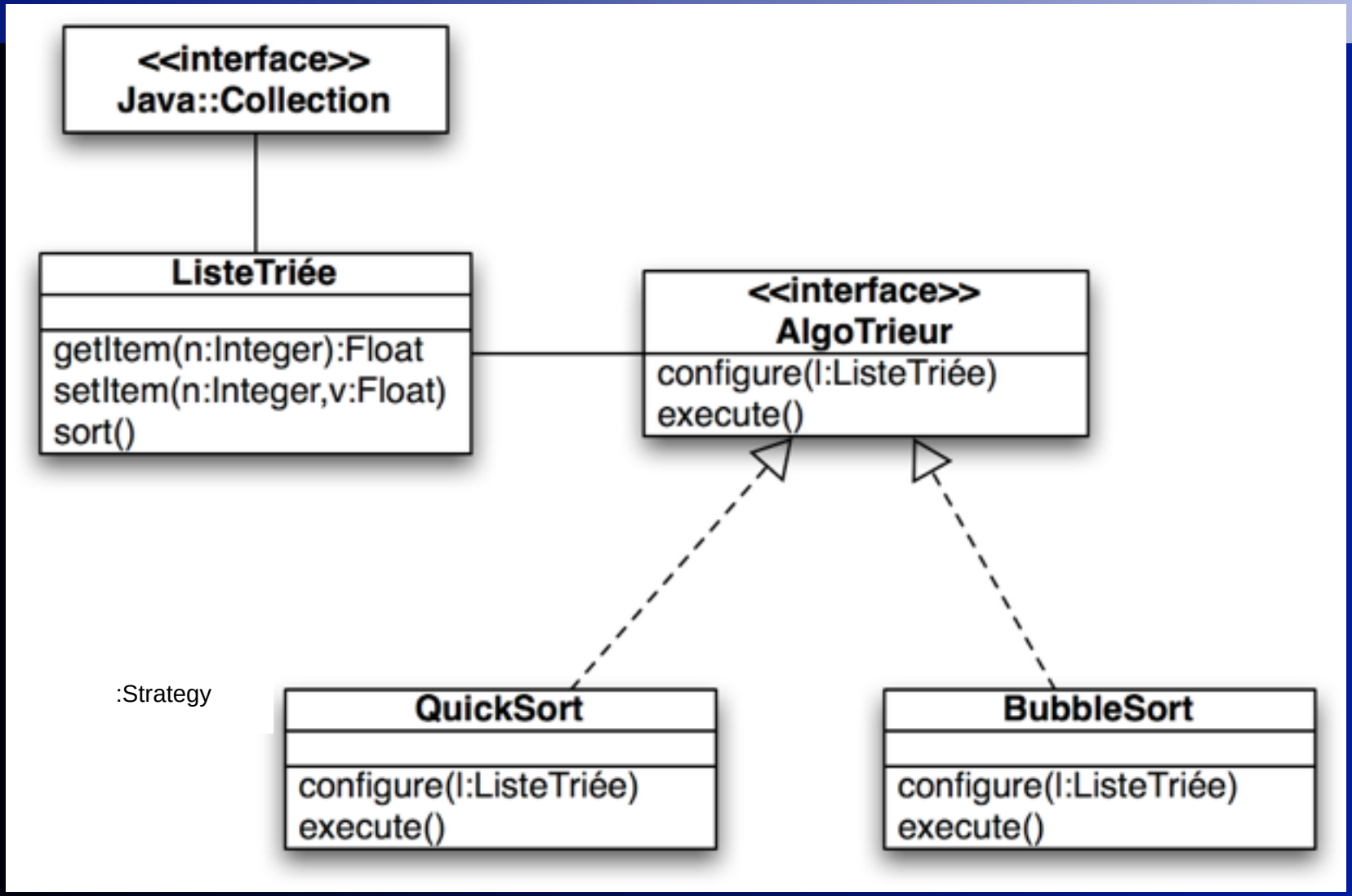
Rôles

- Strategy
  - défnit une interface pour configurer l'algorithme (paramètres) et l'exécuter
- ConcreteStrategy
  - définit une mise en œuvre activée par l'opération d'exécution
- Context
  - désigne l'algorithme concret en vigueur
  - peut contenir des données pour l'algorithme

Diagramme statique



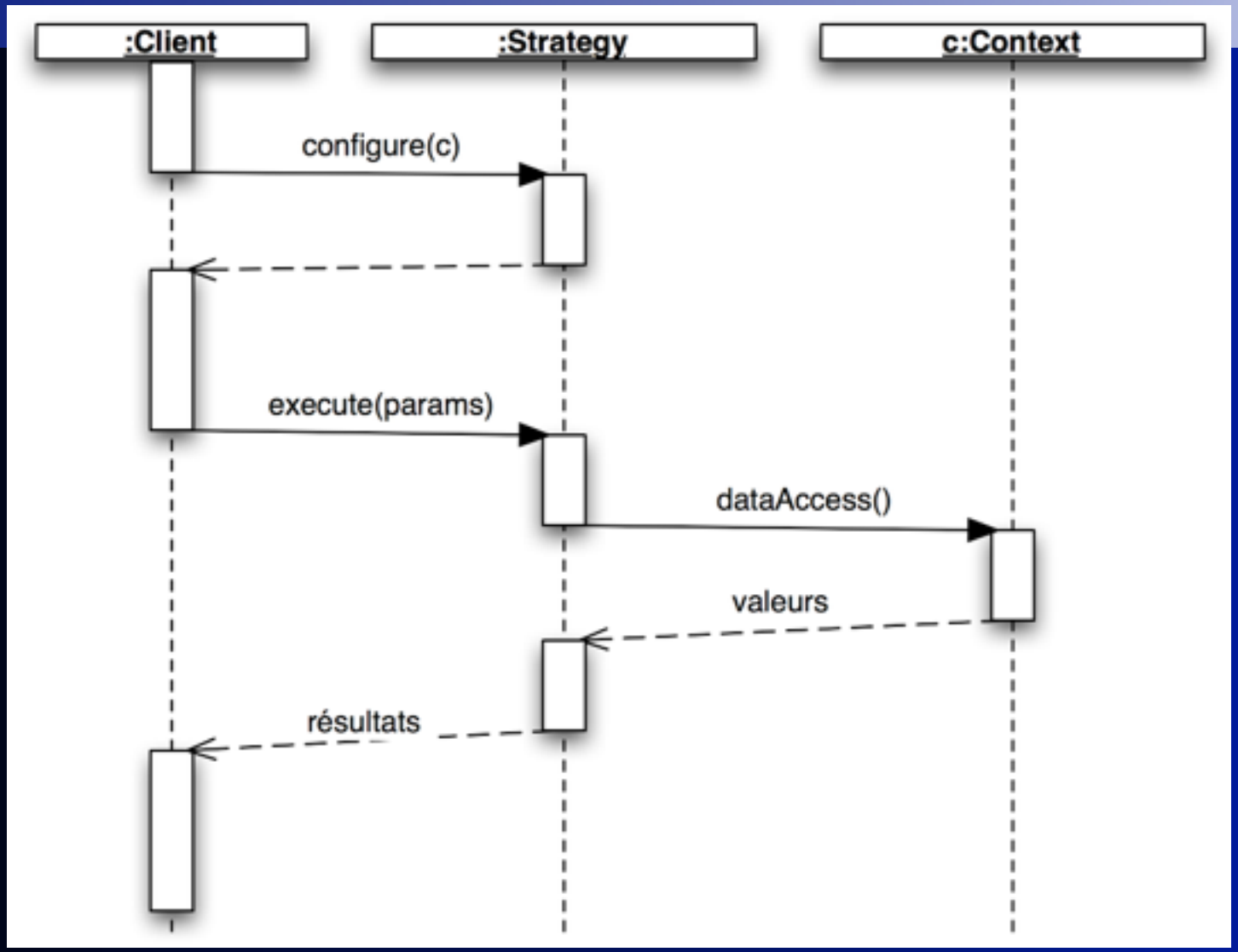
Exemple


<<interface>>
Java::Collection
ListeTriée
getItem(n:Integer):Float
setItem(n:Integer,v:Float)
sort()
<<interface>>
AlgoTrieur
configure(l:ListeTriée)
execute()
:Strategy
QuickSort
configure(l:ListeTriée)
execute()
BubbleSort
configure(l:ListeTriée)
execute()

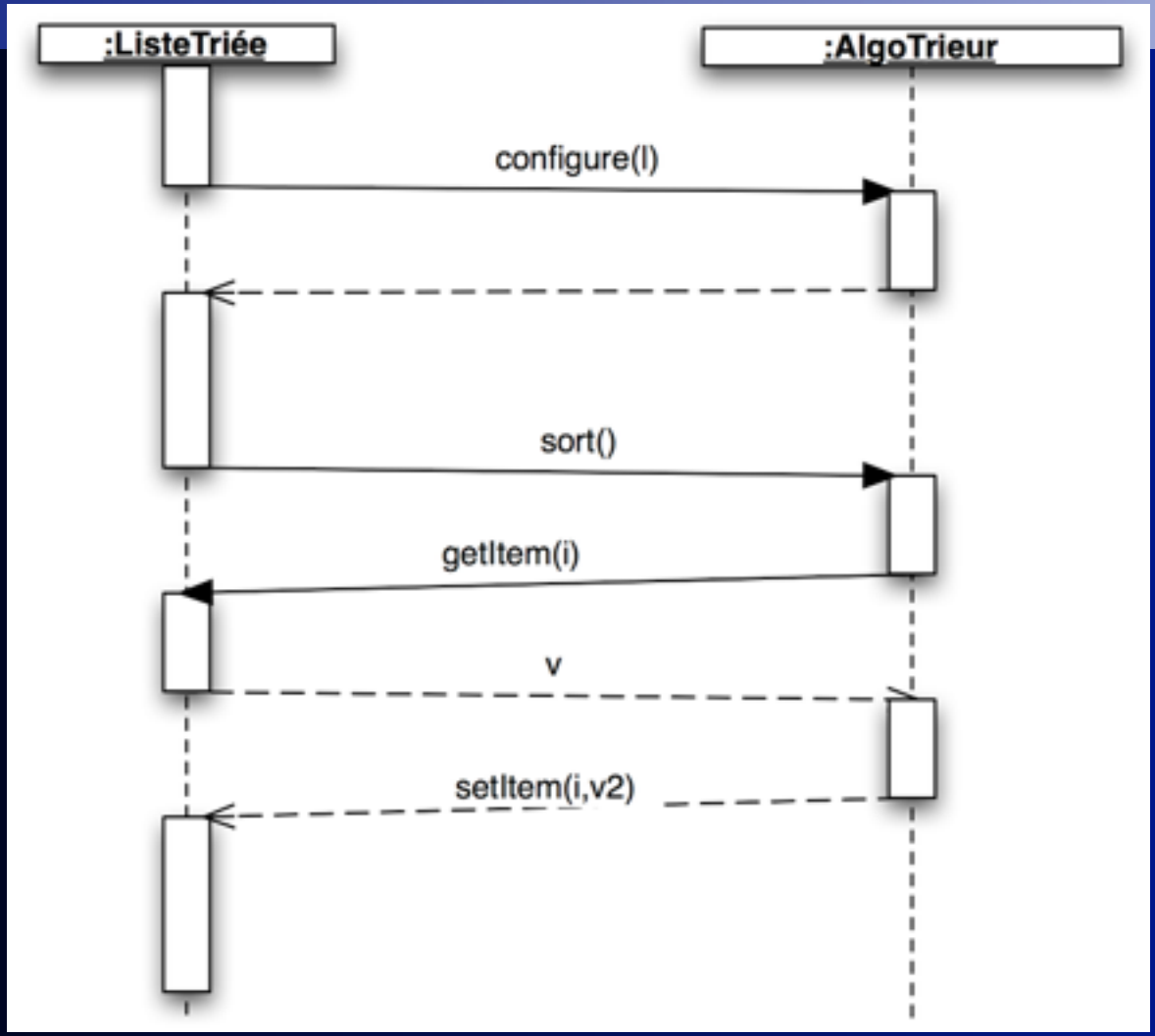
Remarques

- Il est possible de séparer
  - la demande d'exécution
  - la gestion des données
- Dans ce cas on fait apparaître un rôle Client par exemple

Séquence d'exécution avec Client



Exemple de séquence sans Client



Examinons un exemple...

```
// Le rôle de Context

public interface ListeTriee {
   public float getItem(int n);
   public void setItem(int n, float
      v);
   public void sort(); }
```

Le rôle Strategy

```
public interface AlgoTrieur {
    public void configure(ListeTriee l);
    public void execute(); }
```

Le rôle ConcreteStrategy

```
public class BubbleSort implements AlgoTrieur {
    private ListeTriee l;
    public void configure(ListeTriee l) { this.l = l; }
    public void execute() {
    System.err.println("Simulation de tri
    BubbleSort");
} }
```

Une mise en œuvre de Client

```
public class SimpleListeTriee implements ListeTriee {
    private float[] contenu;
    public SimpleListeTriee(int n) {
    contenu = new float[n]; }
    public float getItem(int n) { return this.contenu[n]; }
    public void setItem(int n, float v) { this.contenu[n] = v;
}
```

Suite du client

```
public static void main(String args[]) {
    ListeTriee uneListe = new SimpleListeTriee(10); uneListe.sort(); }
    public void sort() {
        AlgoTrieur unAlgorithme;
        // Choix de l'algorithme
    if (contenu.length < 100) { unAlgorithme = new BubbleSort(); }
    else { unAlgorithme = new QuickSort(); }
    unAlgorithme.configure(this);
    unAlgorithme.execute();
}
```

Des questions sur Strategy ?

EXEMPLES TYPIQUES DE STRATEGY

Stratégies incorporées dans un cycle d'exécution

- Sert au déport de la mise en œuvre dans un autre objet
- Très employée dans certains frameworks (p. ex. Cocoa)
- Une partie de la décision est reportée à l'extérieur
- Analogie avec Template method

Exemple

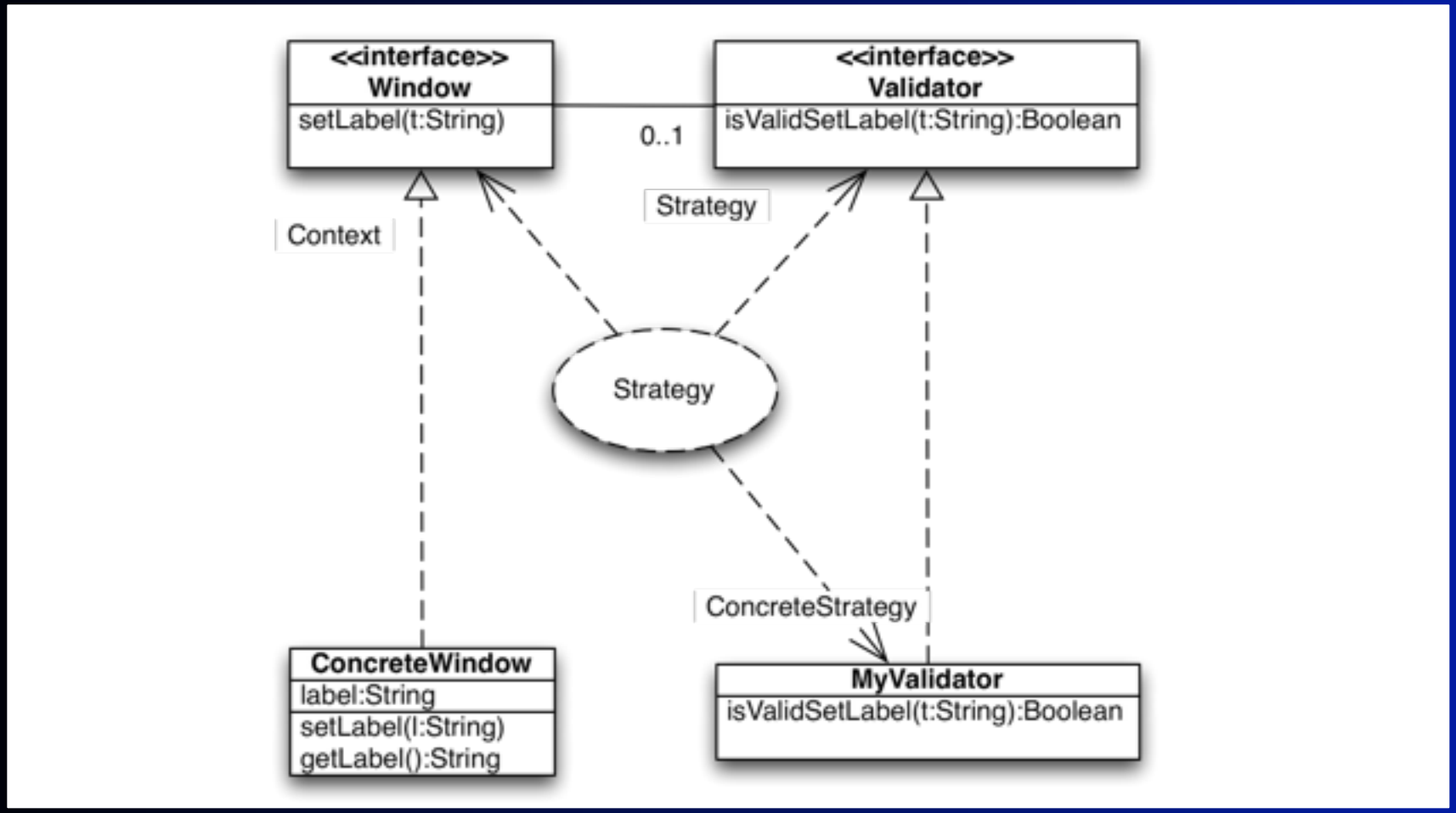
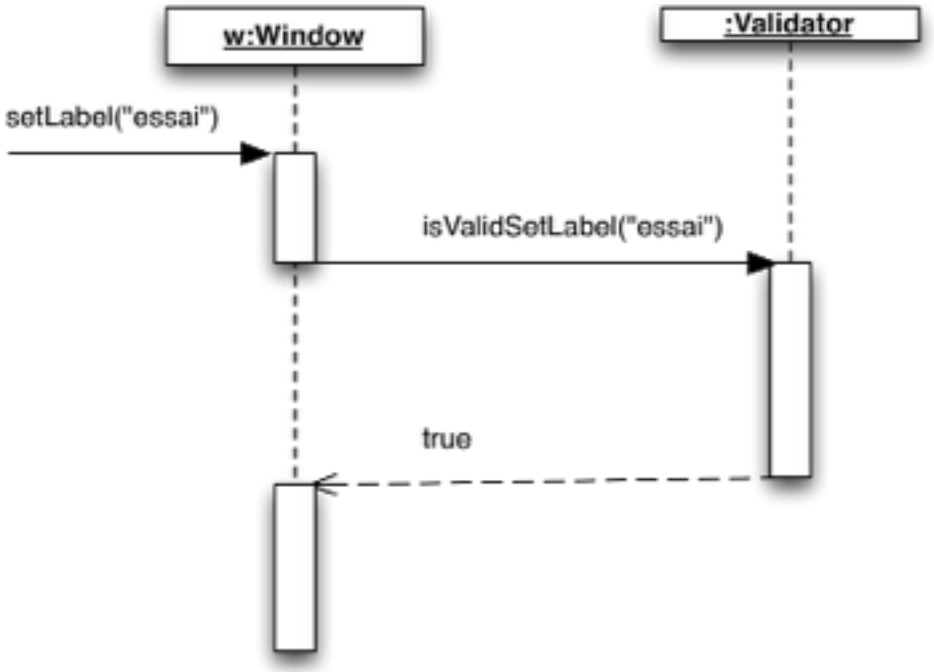

<<interface>>
Window
setLabel(t:String)
0..1
<<interface>>
Validator
isValidSetLabel(t:String):Boolean
Strategy
Context
Strategy
ConcreteStrategy
ConcreteWindow
label:String
setLabel(l:String)
getLabel():String
MyValidator
isValidSetLabel(t:String):Boolean

Diagramme de séquence



Contraintes liées au mécanisme de délégation/proxy

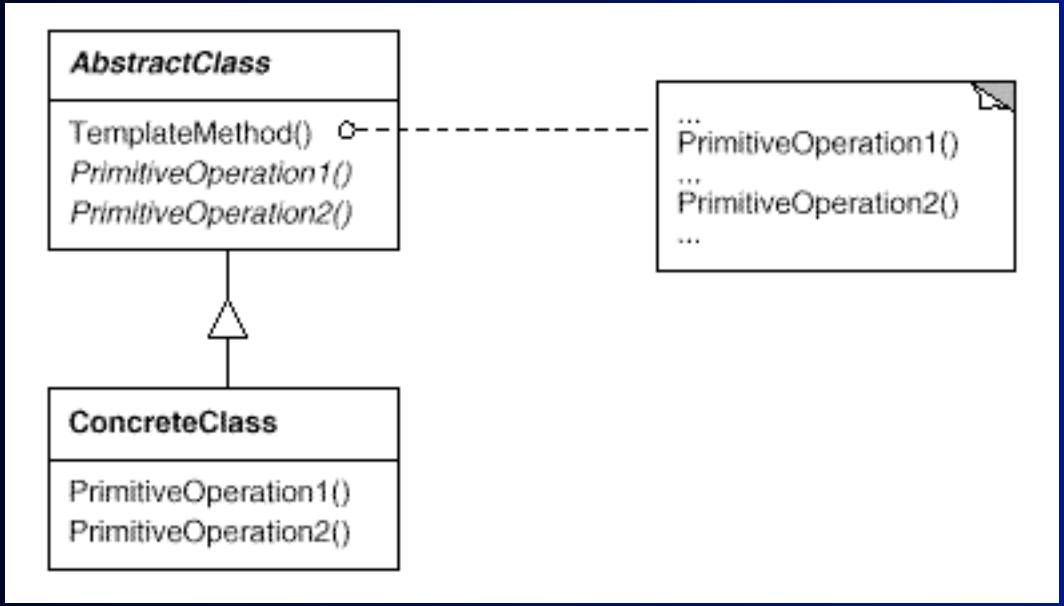
- Nécessité de connaître le protocole du déléguant
- Peut conduire à la multiplication des classes gérant la délégation
- Peut devenir lourd avec du typage statique
  - certains langages permettent des « protocoles informels »
  - un déléguant peut tester l'existence d'opération du délégué (qui n'est donc pas contraint de toutes les mettre en œuvre)

LE PATRON TEMPLATE METHOD

But de Template Method

- Fournir un point d'extension pour une mise en œuvre dans un algorithme
  - par exemple une fonction de comparaison
  - le mécanisme de base est ici l'héritage
- Ce n'est donc pas de la délégation
  - le patron est présenté ici à titre de comparaison

Structure de Template Method



Rôles

- AbstractClass
  - définit un algorithme « à trous »
  - chaque trou est l'appel à une opération interne abstraite
- ConcreteClass
  - étend AbstractClass en fournissant des méthodes pour les opérations abstraites
- Composable avec Strategy

Documentation de l'héritage

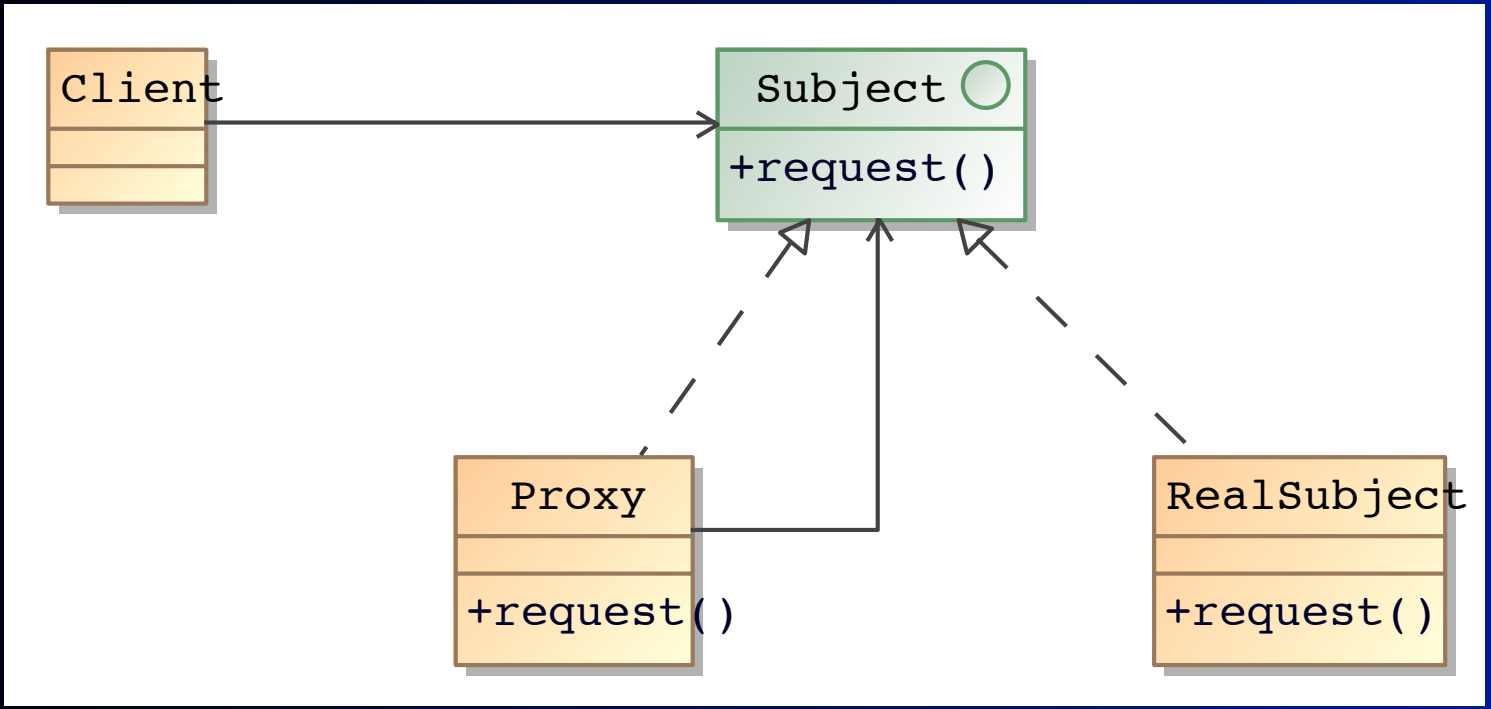
- Comme évoqué dans la partie Strong Objects, l'héritage doit être soigneusement documenté
- Dans Template Method le protocole que chaque sous-classe doit mettre en œuvre est nécessairement explicité
- On pourrait même dire que tout héritage de méthode doit suivre le PC Template Method

LE PATRON PROXY

Buts du pattern Proxy

- Fournir un objet qui agit comme une « doublure »
  - Cette doublure est plus économique/plus simple,...
  - cette doublure a l'interface de l'original
  - elle ne possède que certaines propriétés
  - elle rend les autres services par délégation

Structure de Proxy



Participants

- Subject
  - l'interface de définition du service, des propriétés
- RealSubject
  - le « gros objet » qui détient les vrais attributs et les vraies méthodes
- Proxy
  - fait semblant d'être un gros objet

Avantages de Proxy

- Le principe général permet d'ajouter des services « méta » dans le Proxy
- Exemple : copy on write
  - dupliquer un « gros objet » (RealSubject) duplique le proxy
  - le deuxième proxy référence le premier RealSubject jusqu'à la première modification

Inconvénients de Proxy

- Une indirection peut rendre l'identité de référence incorrecte
  - deux références de Proxy peuvent être différentes et désigner le même RealSubject

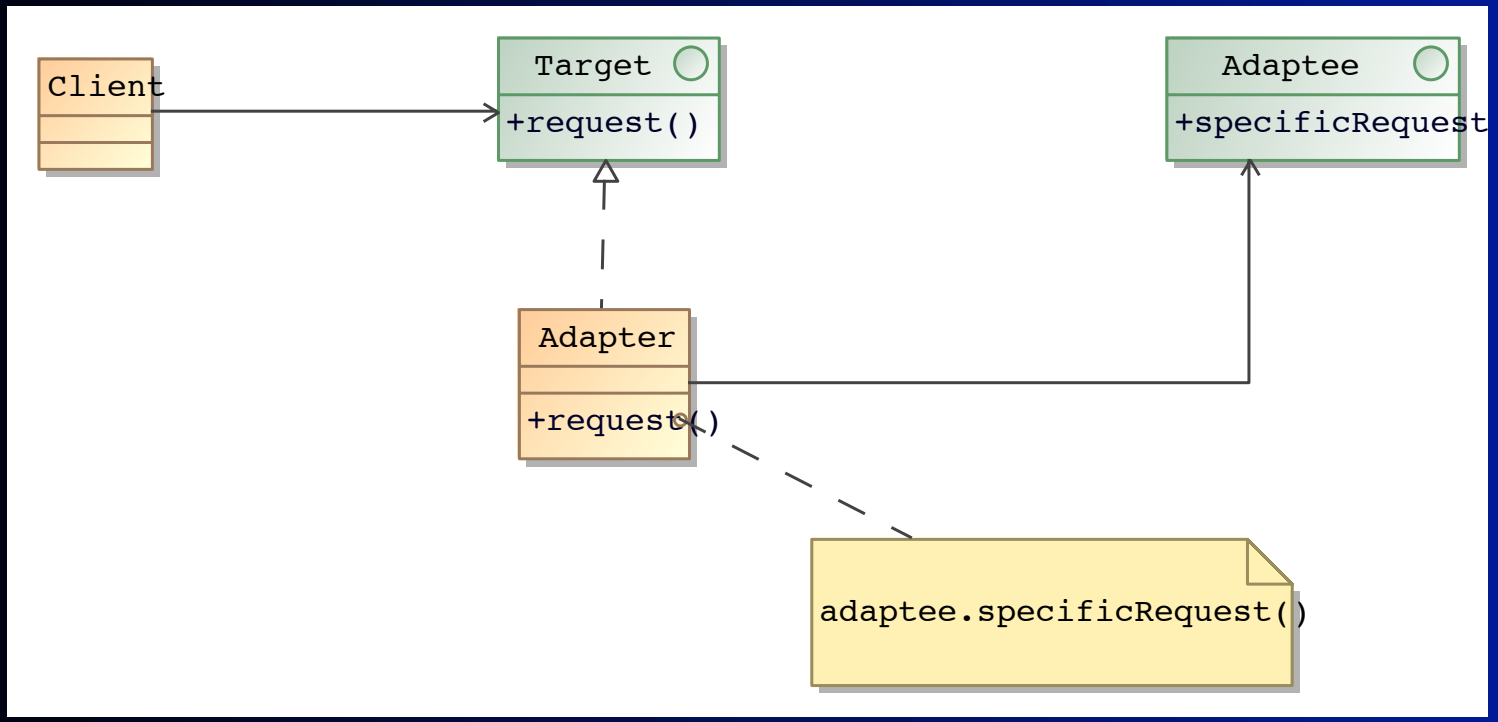
Des questions sur Proxy ?

LE PATRON ADAPTER

But du patron Adapter

- Changer des opérations
  - en réemployant des méthodes existantes
- Catégorie : structure
  - les opérations font en effet partie de la structure
  - à ne pas confondre avec les méthodes

Structure de Adapter



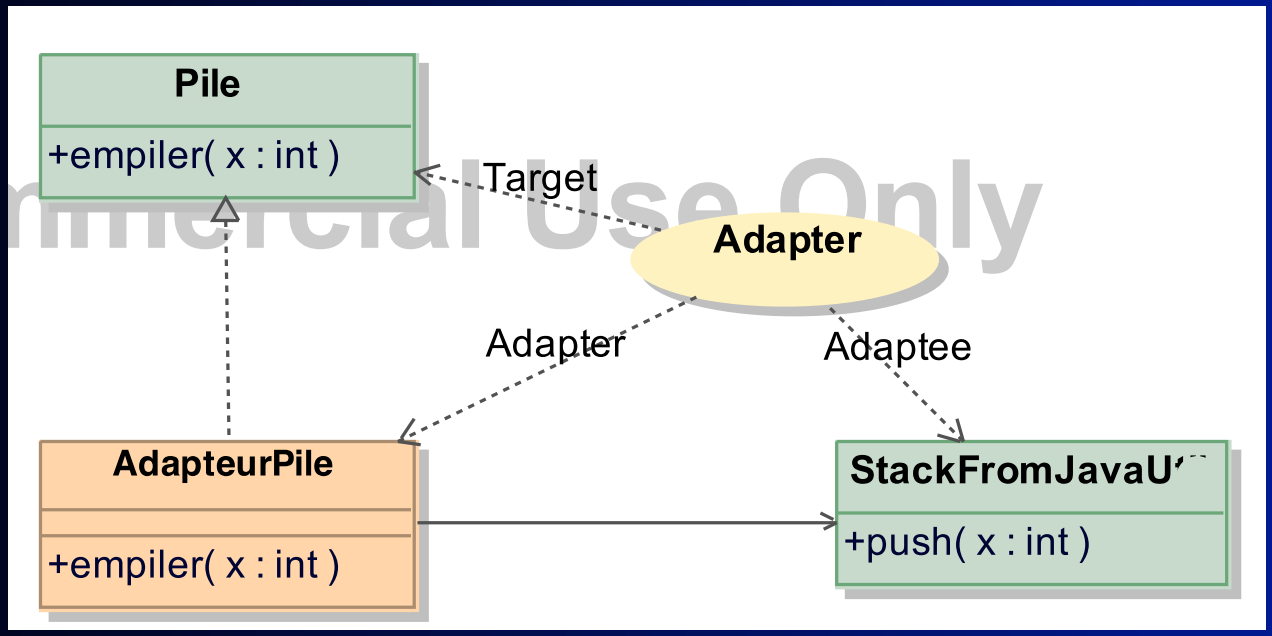
Participants

- Target
  - l'interface qui définit les nouvelles opérations
- Adaptee
  - les anciennes opérations et leurs méthodes
  - accédé par délégation (variante : accès par héritage)

Participants (suite)

- Client
  - accède aux méthodes en employant les nouvelles opérations
  - ignore l'existence des anciennes opérations
- Adapter
  - met en œuvre l'interface Target et réalise les méthodes par délégation
  - effectue toutes les conversions nécessaires

Exemple


Pile
+empiler(x : int)
Target
Adapter
Adapter
Adaptee
AdapteurPile
+empiler(x : int)
StackFromJavaU
+push(x : int)

Des questions sur Adapter ?

LE PATRON DECORATOR

But de Decorator

- Étendre dynamiquement les possibilités d'une objet
  - au moyen de nouvelles méthodes
  - alternative intéressante à l'extension statique de type (héritage)
- Catégorie
  - d'après le GoF : structure
  - en fait il ajoute un comportement (en étendant la structure)

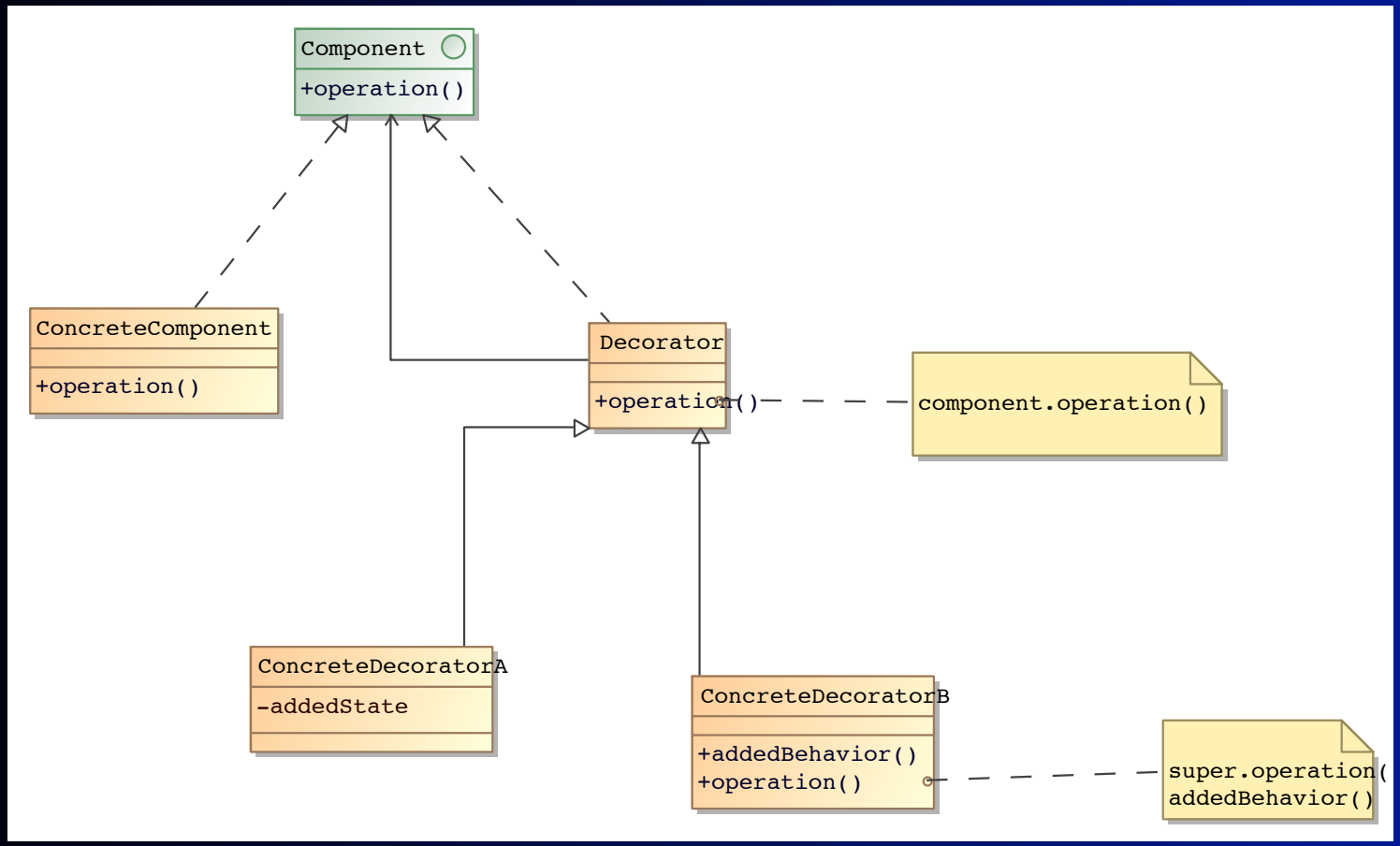
Participants (rôles)

- Component
  - interface définissant les opérations
- ConcreteComponent
  - la mise en œuvre initiale des opérations
- Decorator
  - une classe abstraite qui gère la délégation par défaut

Participants (suite)

- ConcreteDecorator
  - une mise en œuvre étendue par l'ajout d'attributs, de méthodes
  - emploie l'appel à super pour activer la délégation
  - emploie l'appel à des opérations locales pour ajouter du comportement

Structure statique



Exemple

- Decorator dans
  - https://github.com/nplouzeau/dugl

Des questions sur Decorator ?

LE PATTERN FAÇADE

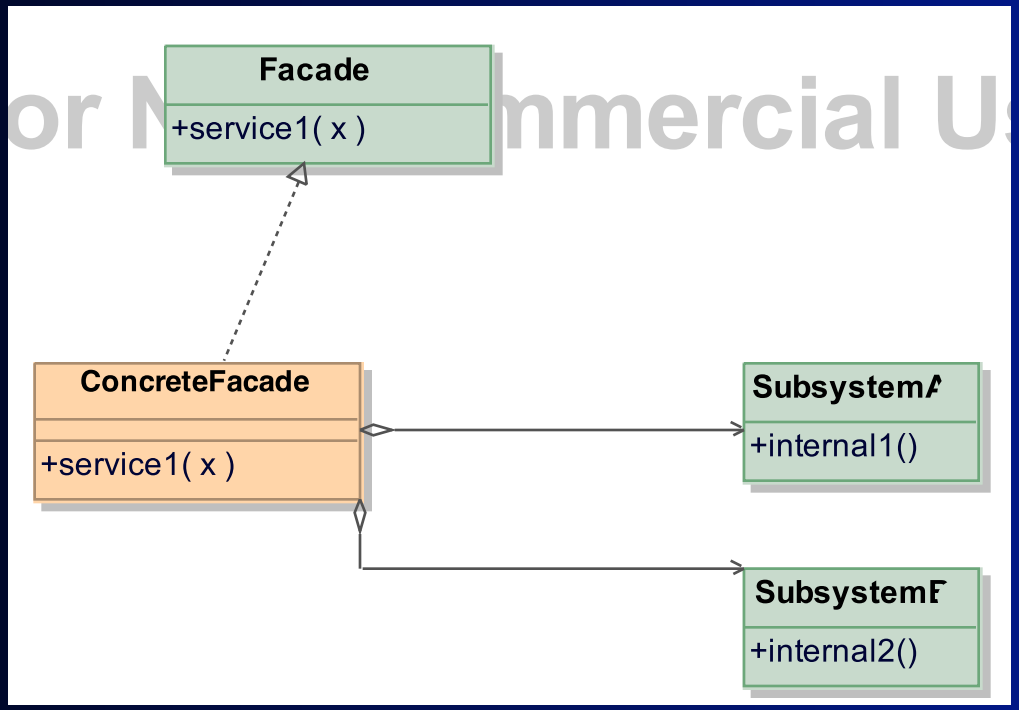
But de Façade

- Rassembler des interfaces en une seule
- Masquer des éléments d’un système

Participants (rôles)

- Façade
  - l'interface qui définit les opérations visibles de l'extérieur
- ConcreteFacade
  - la mise en œuvre de la délégation
- Subsystem
  - les éléments (interfaces, classes) qui composent le sous-système

Structure

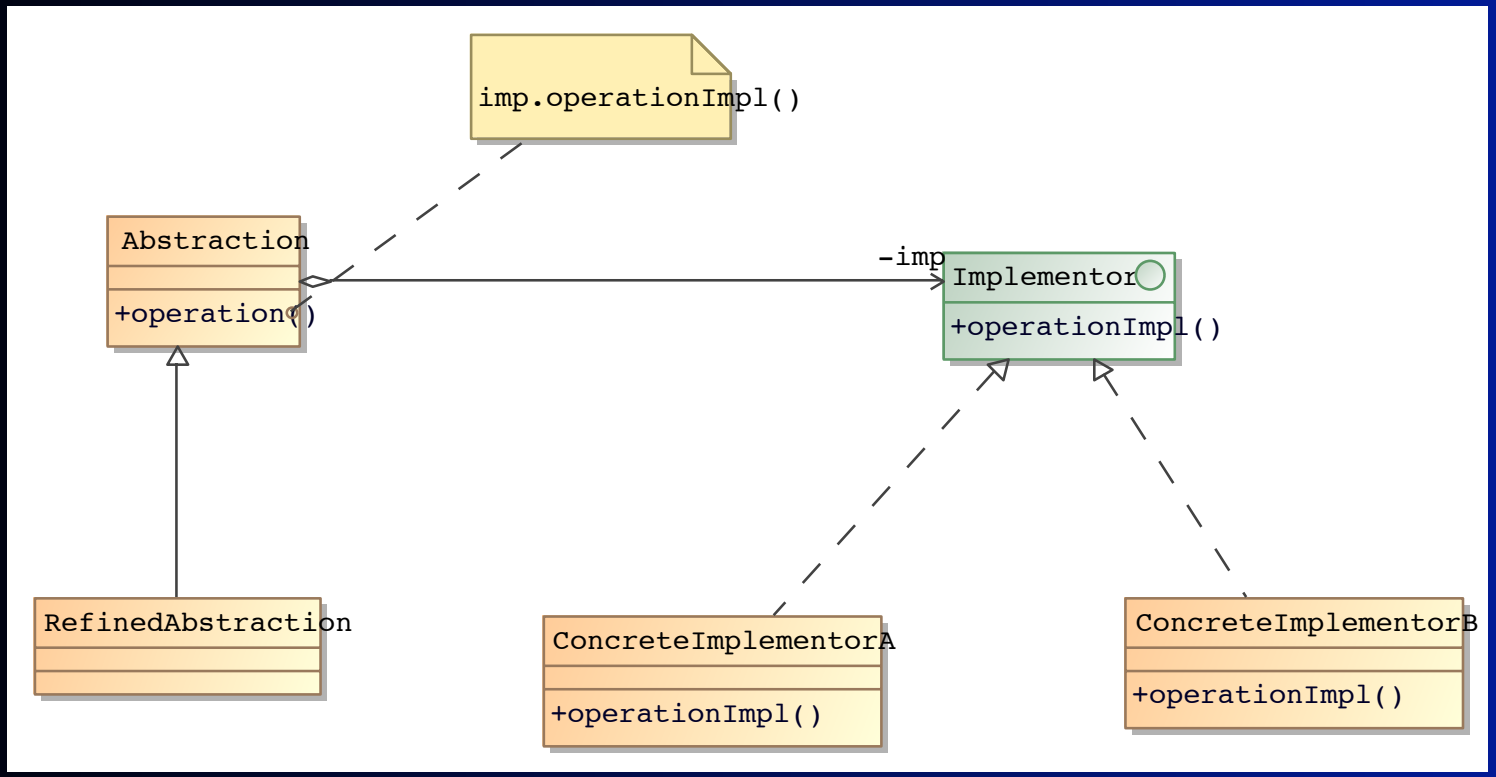

Facade
+service1(x)
ConcreteFacade
+service1(x)
SubsystemA
+internal1()
SubsystemB
+internal2()

LE PATTERN BRIDGE

Le pattern Bridge

- But
  - séparer définition des opérations (interface) et définitions des méthodes en les connectant par délégation
  - permettre une spécialisation séparée des interfaces et des mises en œuvre

Structure de Bridge



Rôles de Bridge

- Abstraction
  - définition des opérations et mise en œuvre du mécanisme de délégation
- Refined abstraction
  - extension des opérations
- Implementor
  - interface pour la mise en œuvre, peut avoir des opérations différentes
- ConcreteImplementor
  - les méthodes mettant en œuvre Implementor

Synthèse des usages

- Emploi de la délégation pour cacher une structure
  - Proxy, Adapter
- Emploi de la délégation pour changer une méthode
  - Strategy, Decorator, Template method, Bridge

LES MÉCANISMES DE DÉLÉGATION AUTOMATIQUE

Principes

- Un objet reçoit un message
- Ce message n'est pas mis en œuvre localement
  - pas de méthode locale ou par héritage de méthode
- Un mécanisme permet de rechercher la méthode
  - exécution d'une opération forwardInvocation

Forward invocation

- Si un objet n'a pas de mise en œuvre pour une opération donnée
  - si il met en œuvre l'opération forwardInvocation alors celle-ci est appelée avec le message
- Invocation
  - contient le destinataire initial, le message et ses paramètres

Exemple noté « à la Java »

- interface Print {
  - File makePDF(String name);
- }
- abstract class SimplePrint implement Print {
  - // makePDF non mise en œuvre
  - void forwardInvocation(Invocation i) {
    - unDelegue.do(i)
- }

Suite exemple

- Print test = new SimplePrint();
- File f = test.makePDF(“Test”);
  - // En Java ne passerait pas à la compilation
  - // Provoque
    - la création d’un objet Invocation i
    - l’appel de l’opération forwardInvocation(i)
  - // l’appel de la mise en œuvre de forwardInvocation

Limites de la technique

- Nécessité de pouvoir manipuler les invocations comme des objets
  - réification
  - mots clés spécifiques (DSL) pour alléger
- Typage dynamique sous contrôle du développeur
  - tendance actuelle due aux architectures dynamiques