

Design Patterns

Matthias Colin

Programme

- Concepts POO
- Design Patterns
- Application en Java

Historique

- ***Design Patterns: Elements of Reusable Object-Oriented Software*** par le gang of 4 (gof)

Erich Gamma, Richard Helm, Ralph Johnson et John Vlissides (1994)

- 23 Problématiques avec solution
 - 1 version générique et 1 exemple concret
 - Modélisation en “UML”
 - Solution en C++ et/ou smalltalk
- Adaptation en Java

UML

- statique:
 - diagramme de classes
- dynamique
 - diagramme de séquence ou collaboration
 - diagramme d'états transitions

Classification des Design Patterns

- construction
 - singleton
 - fabrique abstraite, méthode de fabrication
 - monteur(builder)
- structures de données
 - composite
 - décorateur
 - adaptateur, façade
- comportements
 - itérateur
 - état
 - observateur
 - commande
 - template, strategy, visiteur

Patterns de fabrication

Problématique: appel à un constructeur => code peu maintenable

```
Airbus380 plane = new Airbus380(couleur, nombreSiege)
```

comment passer à un Airbus320 ?

Principe de substitution de Liskov **LSP** mit en échec par l'appel au constructeur

Exemples en Java

- List
- String
- objet custom Plane avec plusieurs attributs

Fabriques

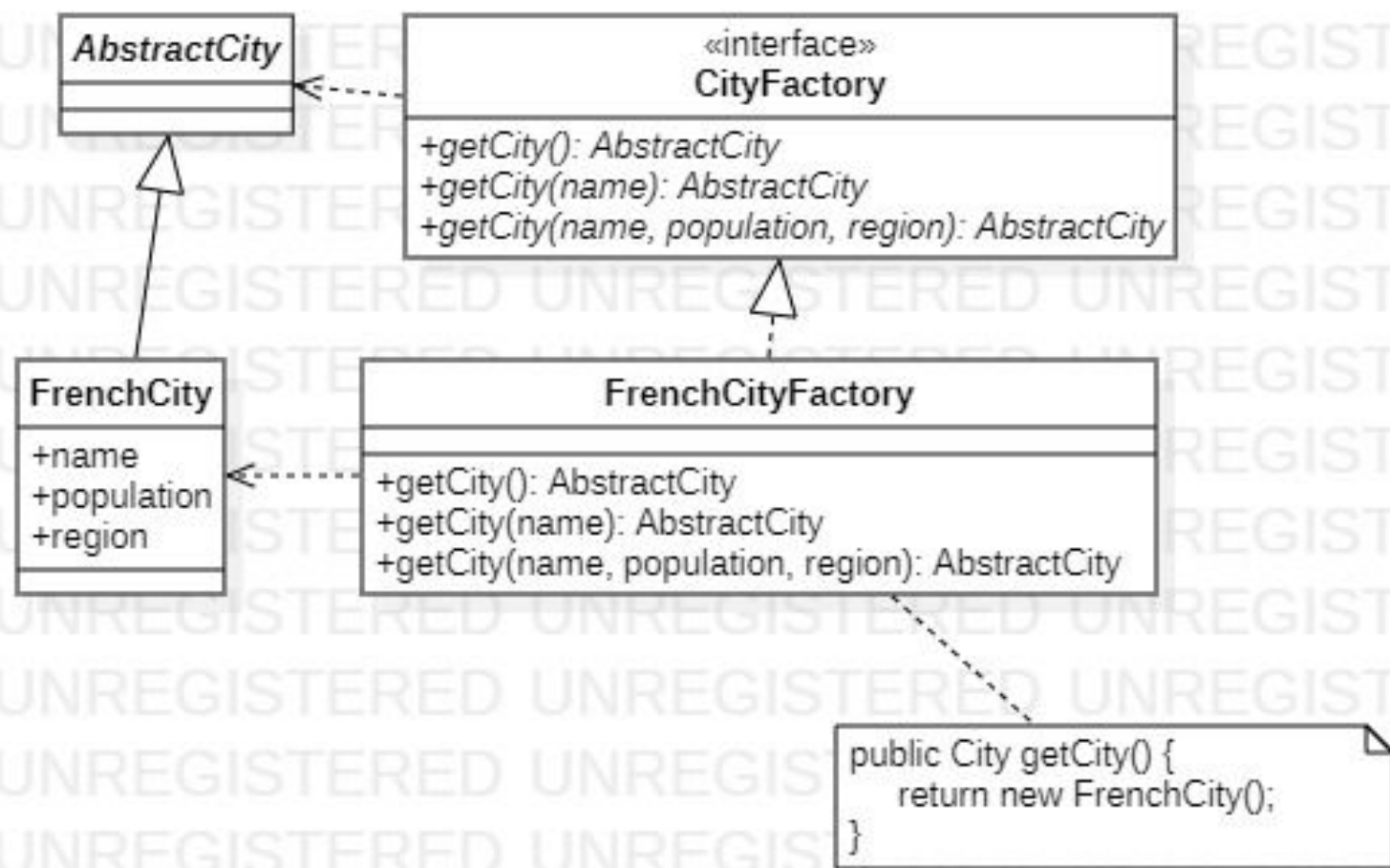
Objectif: encapsuler l'appel d'un constructeur (new) dans une méthode

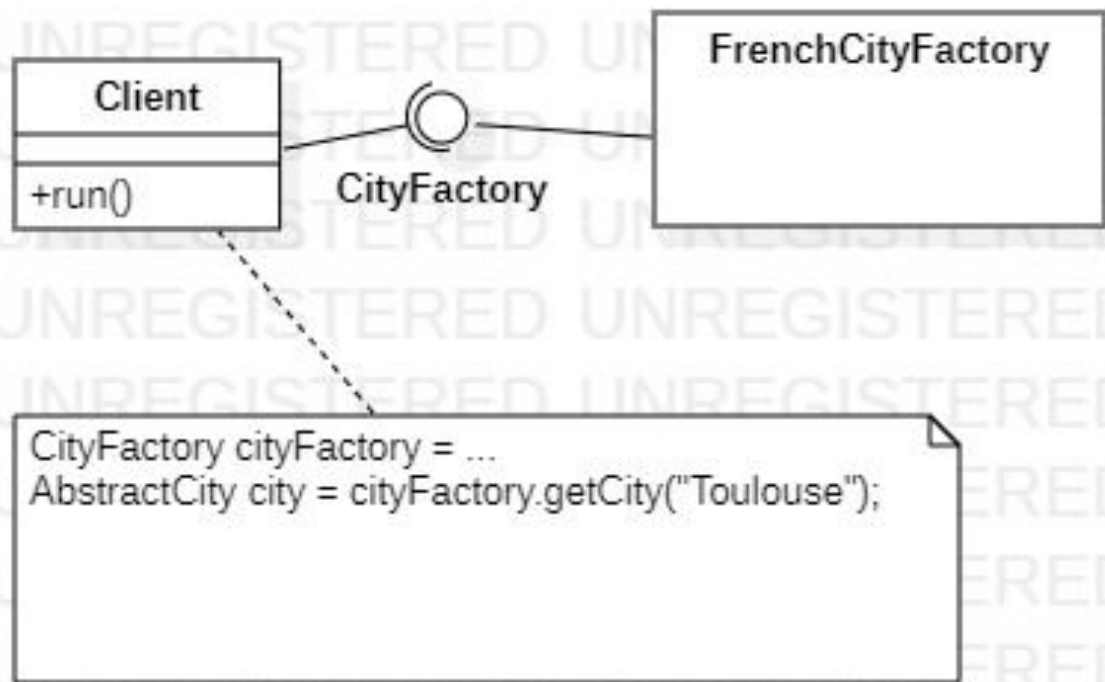
2 patterns:

- Factory method (Fabrique, Fabrication, Méthode de fabrication)
 - 1 seul type d'objet (Ex: Plane)
- Abstract Factory (Fabrique abstraite)
 - 1 famille d'objets cohérents (Ex: Mobilier => Table, Chaise, Lampe, ..)

Eléments de solution:

- classe de l'objet à fabriquer => niveau d'abstraction Interface ou class abstraite





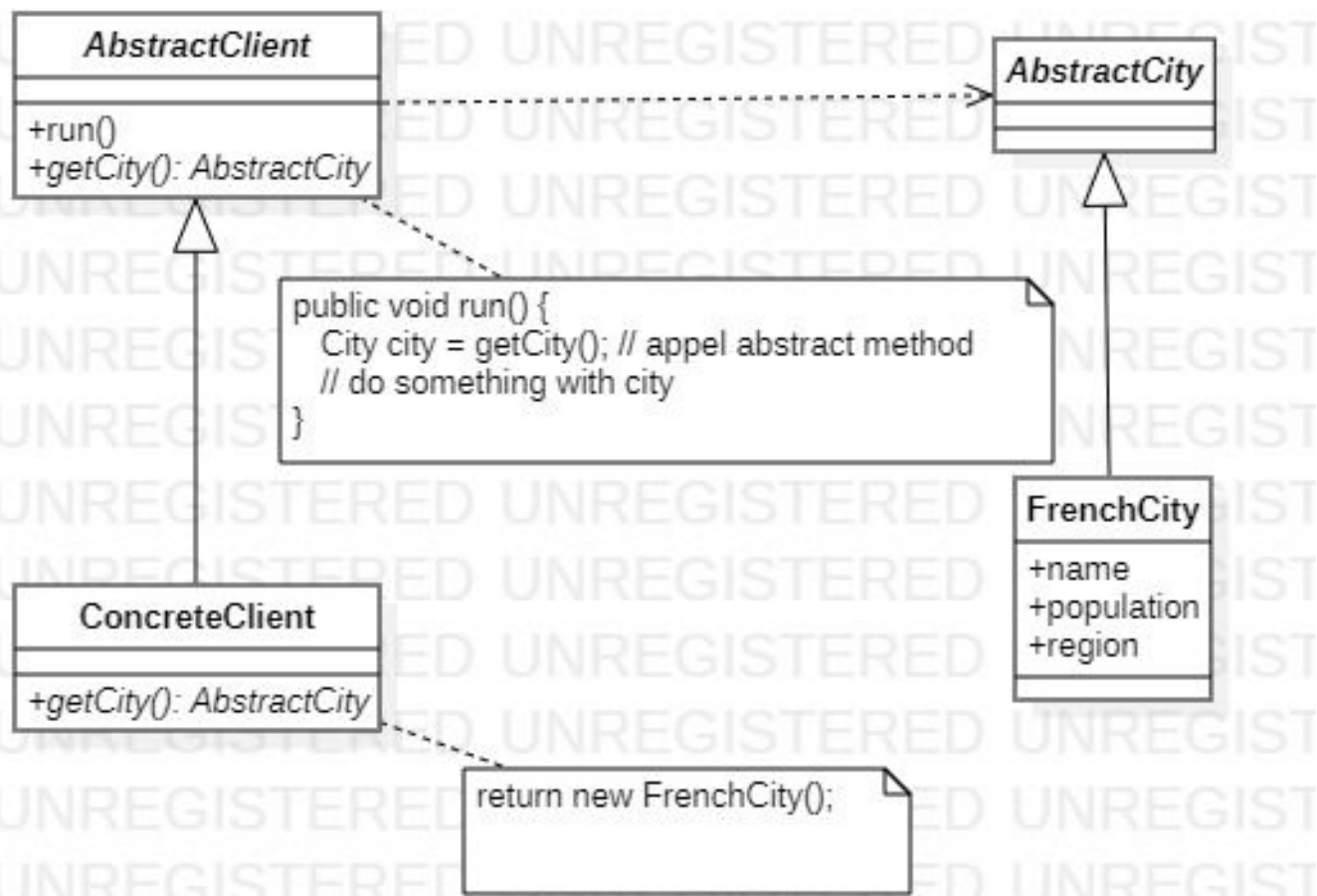
Problématique: comment obtenir l'instance de la fabrique

- fabrique par défaut (singleton)
- configuration externe (Injection dépendance)

Simplifications

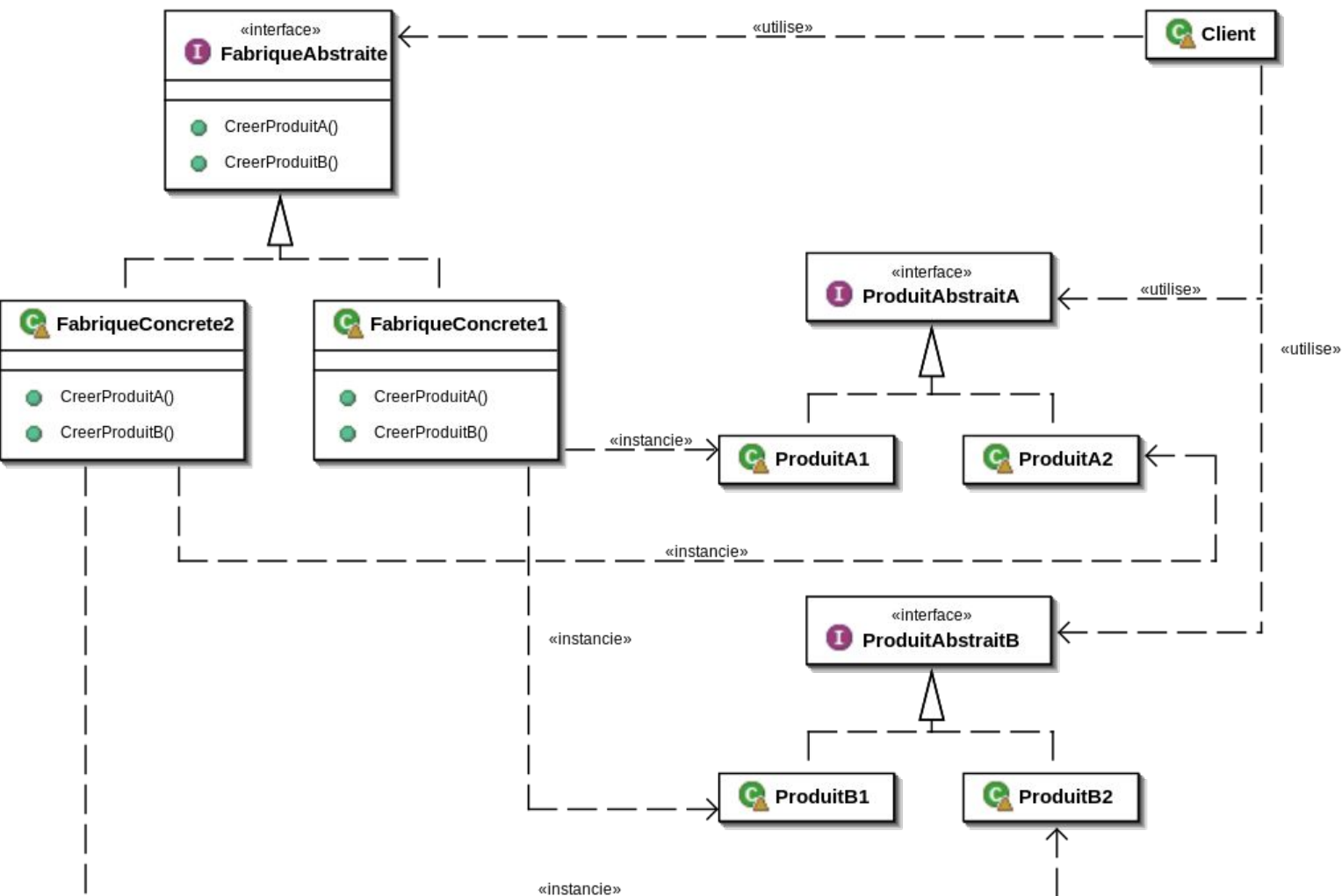
- Classe abstraite du produit est fusionnée avec la fabrique
 - `List.of(...)`
 - `Stream.of(...)`
 - `LocalDatetime.of(year, month, day)`
 - `LocalDatetime.of(year, month, day, hour)`
 - `LocalDatetime.of(year, month, day, hour, minute)`
 - `Optional.of(city), Optional.empty()`
- Classe client et classe Fabrique fusionnées

(cf slide suivant)



Extensions

- Abstract Factory (Fabrique Abstraite)
 - Plusieurs type de produits => une méthode de fabrication par type de produit
 - Exemple: en XML, on peut avoir des éléments, textes, commentaires, attributs,
 - interface Document est une fabrique abstraite avec les methodes
 - Element createElement(...)
 - Text createNodeText(...)
 - Attr createAttribute(...)



Principes Objets

- substitution de LSP
- separation of concerns SoC (Fr: responsabilités, préoccupations)
- Injection of dependency IoC

Singleton

- Problématique: unicité d'une instance d'une classe particulière
- Exemples:
 - Factory
 - Connection ou DataSource
- Solution
 - constructeur privé remplacé par une méthode de classe publique
 - méthode de classe contrôle unicité
 - attribut de classe privé pour mémoriser l'instance unique
- Discussion:
 - passage de paramètre ?

Adaptateur / Adapter

Problématique: adapter un besoin nouveau à un existant avec une interface légèrement différente (nom de méthode, paramètres)

Intérêt: réutilisabilité

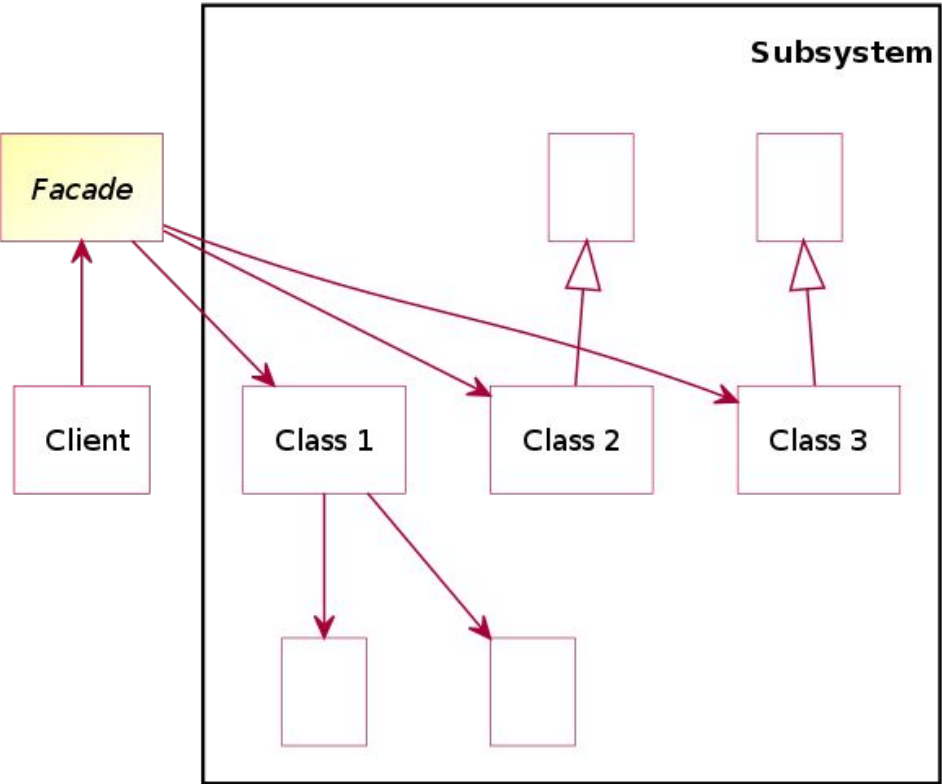
Solutions: héritage vs **composition** (+ souple, adaptable)

Extensions: Façade

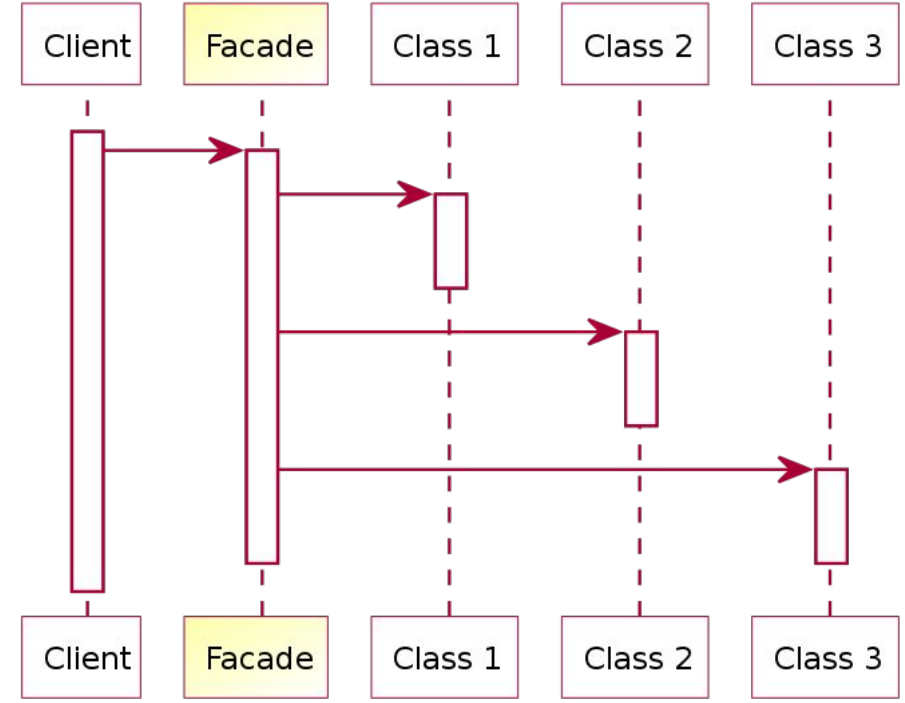
Façade

Avantages:

- maintenabilité
- indépendance du code client vis à vis des classes derrière la façade
- simplifier des api complexes en une api correspondant au besoin



Sample class diagram



Sample sequence diagram

source Wikipedia: https://en.wikipedia.org/wiki/Facade_pattern

Composite

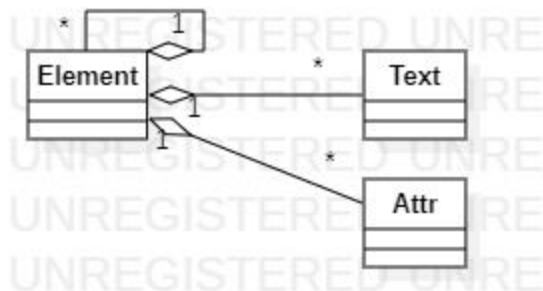
Problématique:

- structure arborescente: xml, json, filesystem, ihm
- propager une opération dans l'arborescence
 - Exemple: taille totale du filesystem, pretty print xml, ...

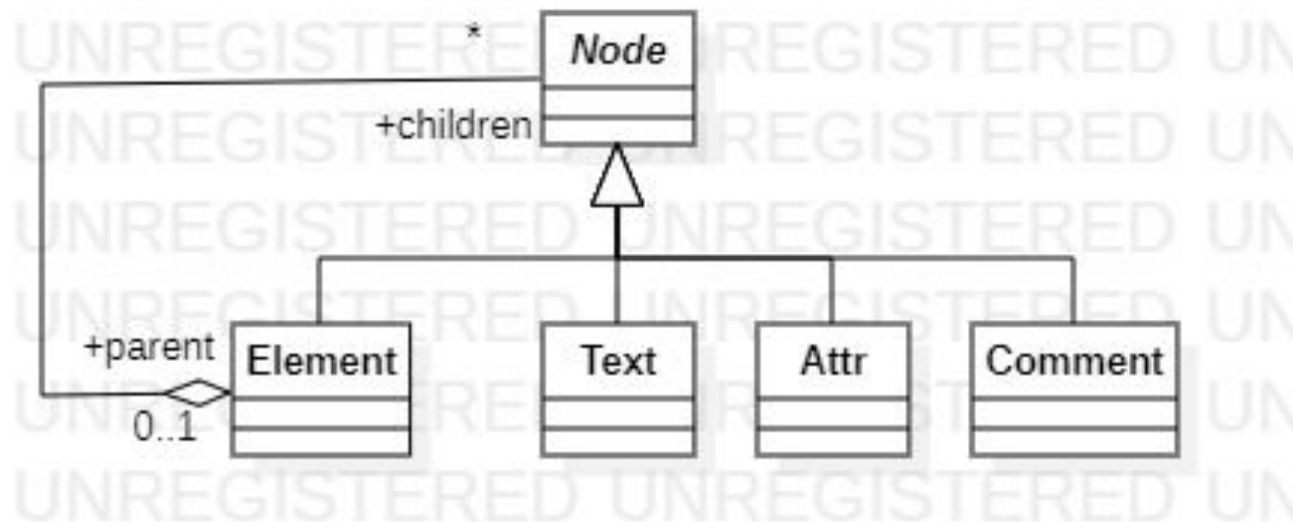
Solution:

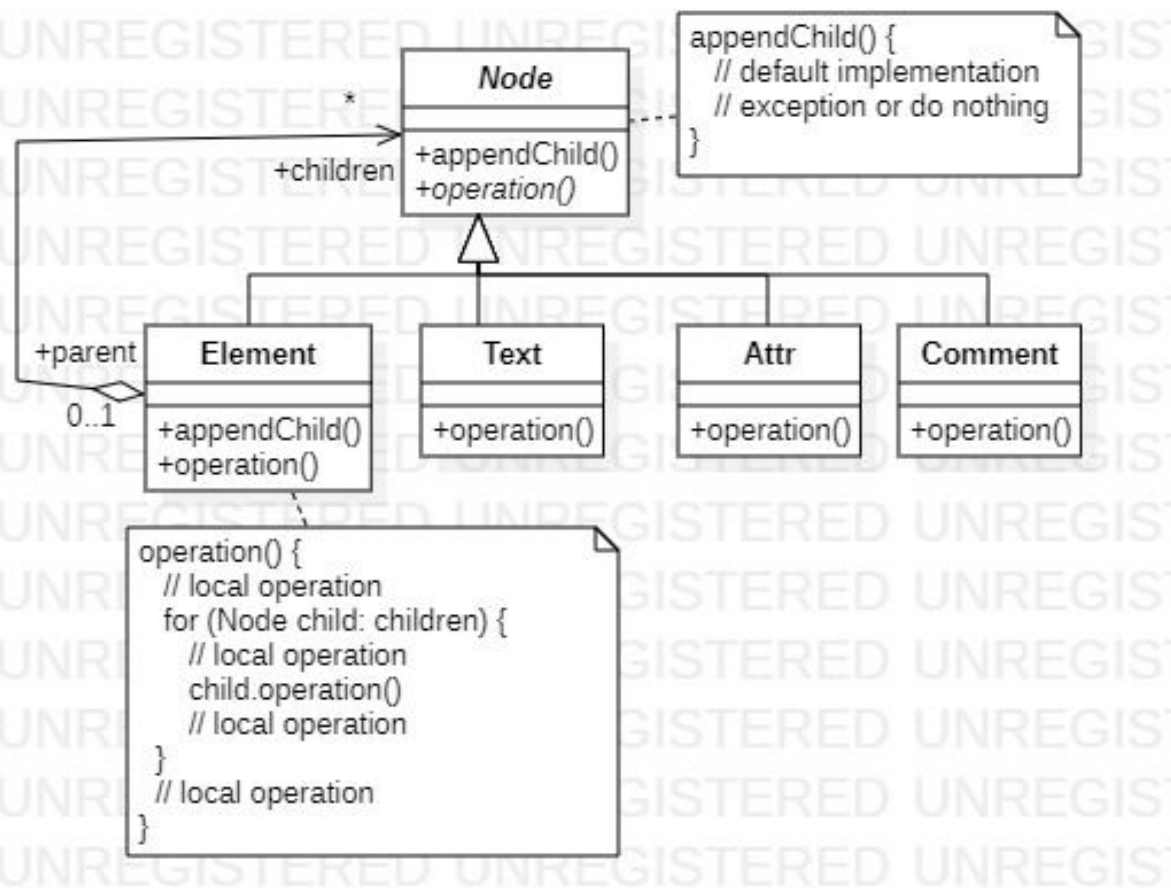
- classes:
 - Composite: conteneur
 - Leaf: feuille(s)
 - *Component* : classe abstraite ou interface de généralisation
- méthodes de gestion des enfants au niveau de Component (+ souple)

Solution bancaire



Solution du pattern

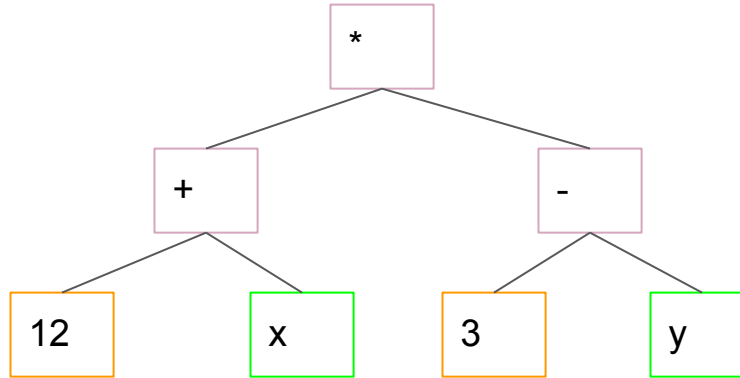




Atelier Composite

- Expression arithmétique
- **Opérateur**: +, -, *, ...
- **Valeur** numérique (float)
- **Variable** (x, x1, y)
- Opération: toString

Exemple: $(12 + x) * (3 - y)$



Itérateur / Iterator

Problématique: parcours d'une structure de données (tableaux, listes, ensembles, listes d'associations, graphes, ...)

Solution:

- externaliser le parcours
- chaque itérateur est spécifique à la structure qu'il parcourt
- la structure délivre son itérateur



Implémentation dans les langages

- Java: interfaces Iterable (iterator()) et Iterator (hasNext(), next())
- Python: objet iterable (__iter__) et iterator (__next__)
- C++: structure fournit 2 itérateurs (begin(), end()), itérateur(++ , *, ==)

Bonus: boucle “foreach” utilisant implicitement l’itérateur

for (Plane plane: planes) { ... } // Java, C#, C++

for plane in planes // python, delphi

for plane of planes //javascript

Strategy

Problématique: algorithme avec **coût** important disponible en plusieurs variantes

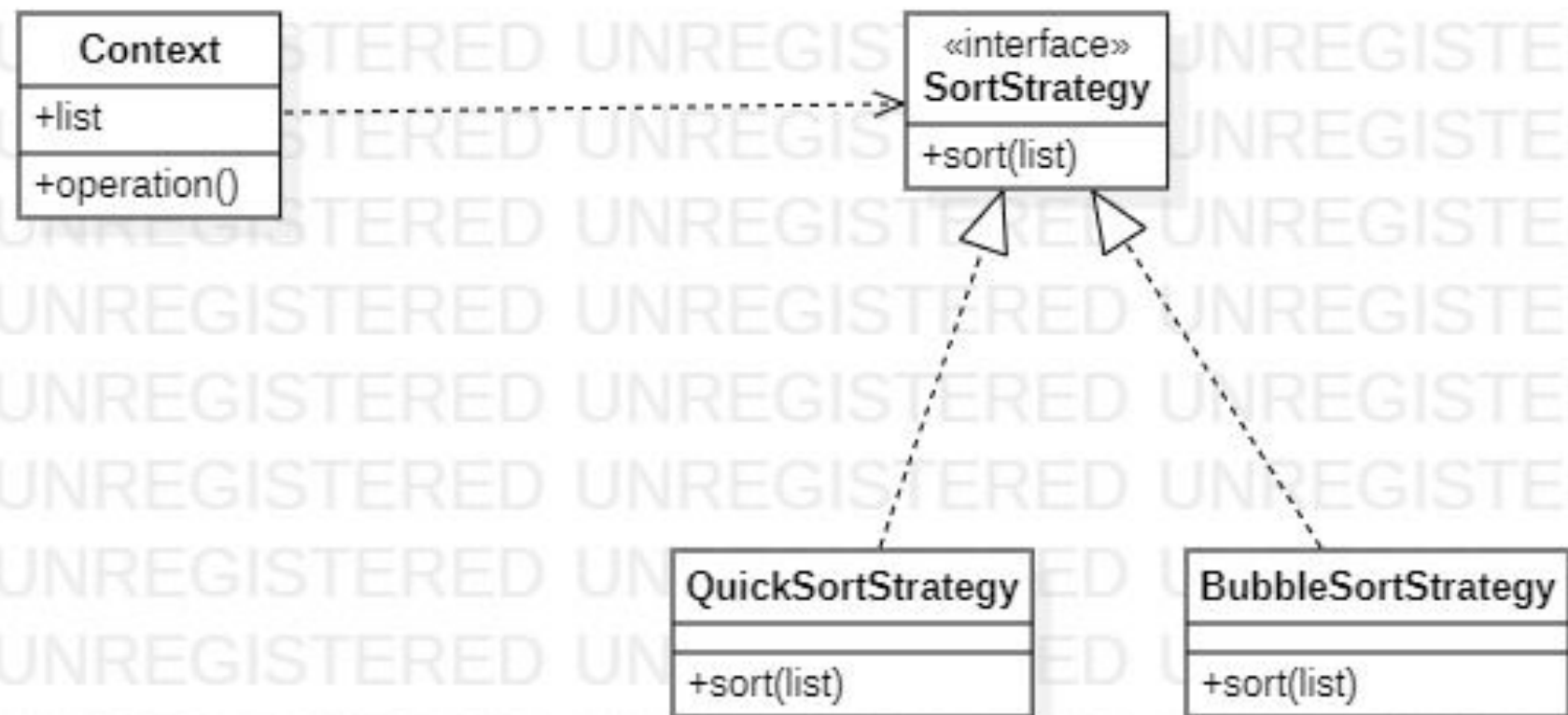
Exemple: tri => à bulle, insertion, rapide, fusion

Exemple: base de données, SELECT

- ... where id = 125 => stratégie: parcours par l'index associé à la PK
- ... where region is null; => stratégie: balayer toute la table

Choix de la stratégie:

- constructeur du contexte
- setter dans le contexte
- paramètre de l'opération
- choix dynamique dans le contexte



sd SeqOperationChoosingSortStrategyDynamically



: User

: Context

: QuickSortStrategy

: BubbleSortStrategy

1 : operation

alt ChooseStrategy

[list.size() >= 10]

2 : sort(list)

[else]

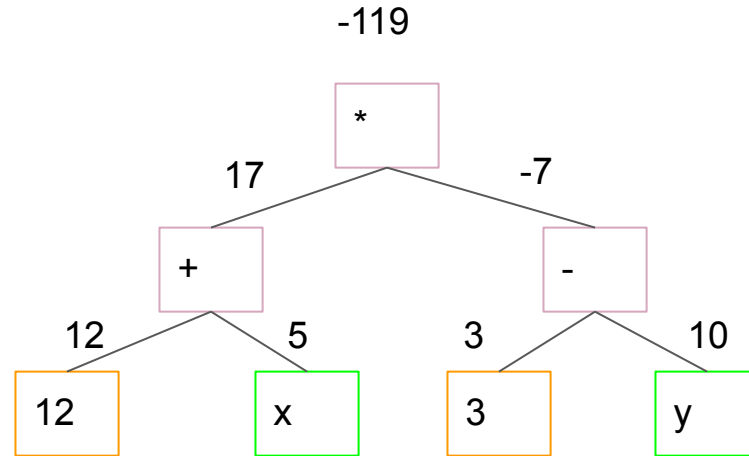
3 : sort(list)

Atelier Composite + Visitor

Valeurs des variables pour évaluation

$x = 5$

$y = 10$



Observateur / Observer

Problématique: un objet observé change d'état et un certain nombre d'observateurs veulent être au courant de ce changement pour réagir (rafraîchissement, synchro base de données, ...)

Commande / Command

Problématique: couplage faible entre un donneur d'ordre et un executeur (receiver)



sd SequenceDiagram1

jbtOk: JButton

: CommandOk

: MainWindow

city: City

: CityViewPanel

1 : actionPerformed

2 : updateModelFromViewPanel

3 : getCity

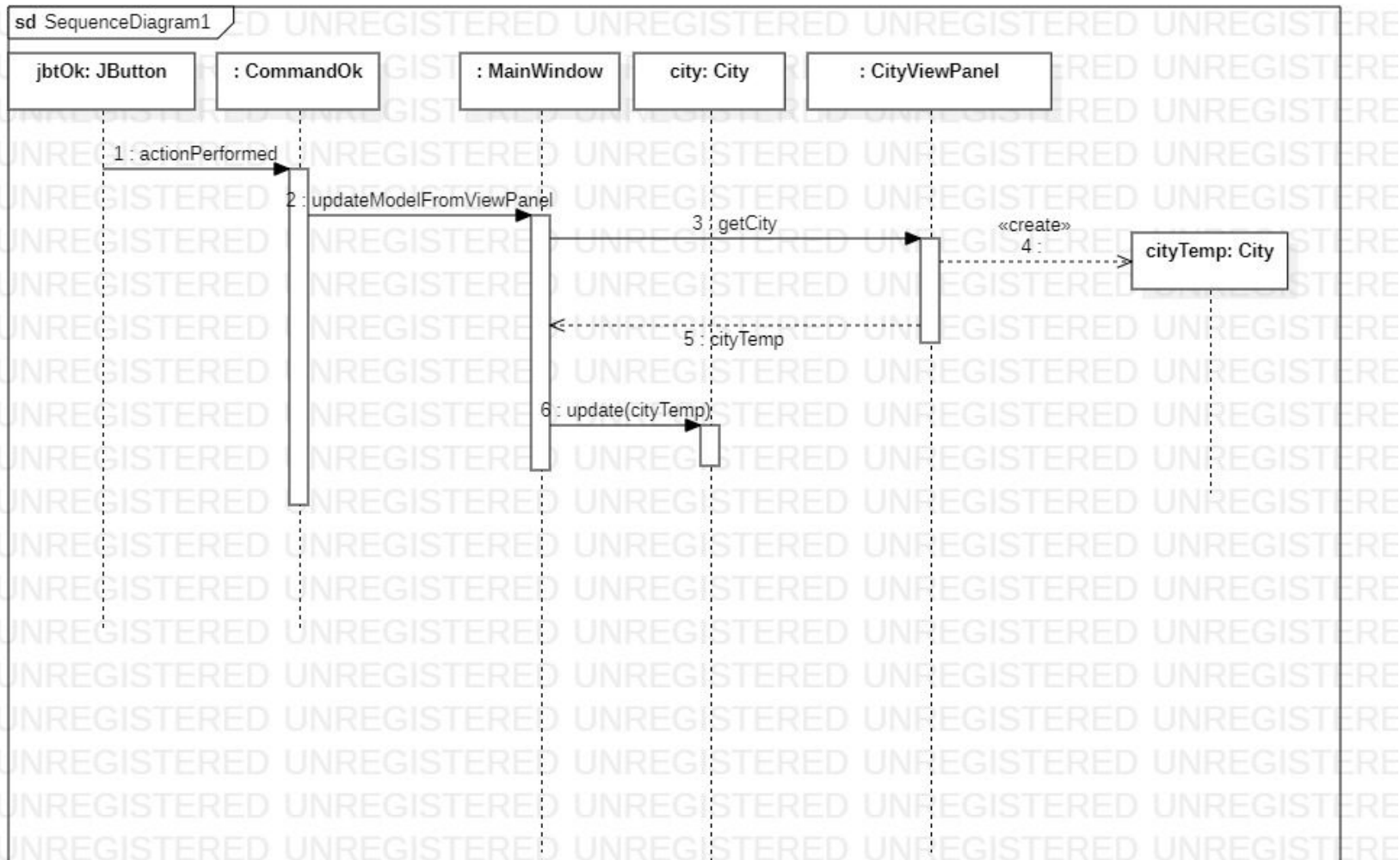
«create»

4 :

cityTemp: City

5 : cityTemp

6 : update(cityTemp)



IoC

IoC: Inversion of Control => DI: Injection de dépendance

Branchement d'une classe concrète derrière une interface

Exemple:

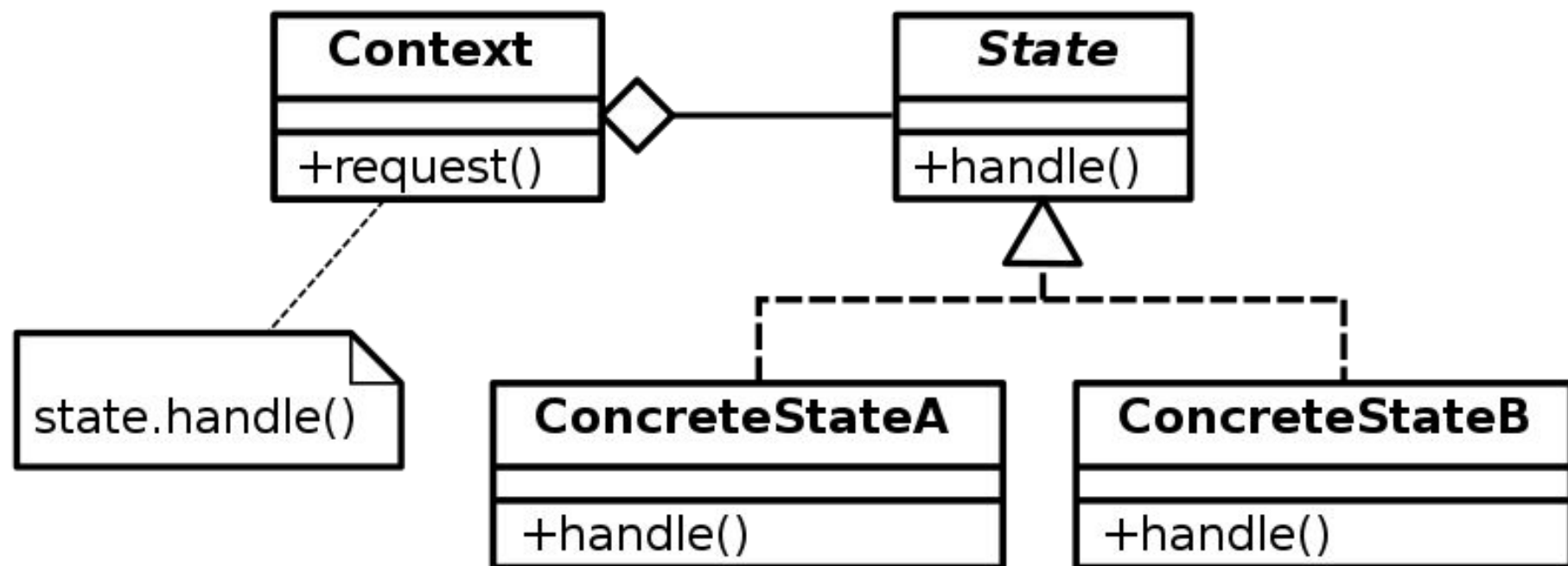
- Springboot: @Autowired
- Asp.net : configuration class

Etats Transitions / State

Problématique: un objet à plusieurs états, i.e ayant un comportement différent en fonction de son état.

Exemple: distributeur bancaire, barrière de péage

Solution: externaliser le traitement des fonctions dans des classes Etats



/ fondDisponible = 0

[fondDisponible = 0]

HorsService

alimenter / fondDisponible += somme

EnService

introduireCarte / essai = 0

CarteIntroduite

taperCode / essai++

[code incorrect et essai < 3]

[code incorrect et essai = 3] / avalerCarte

[code correct]

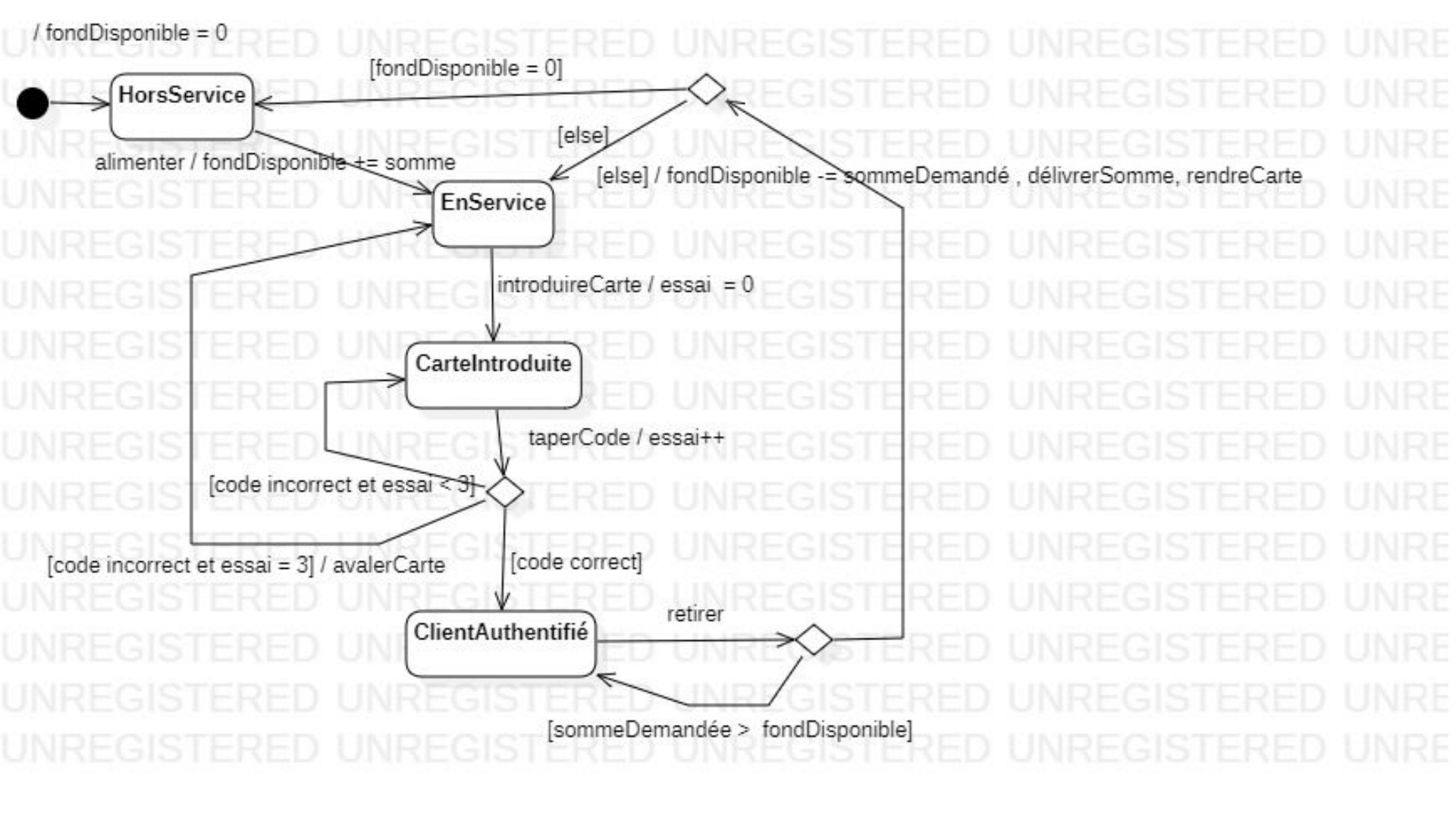
ClientAuthentifié

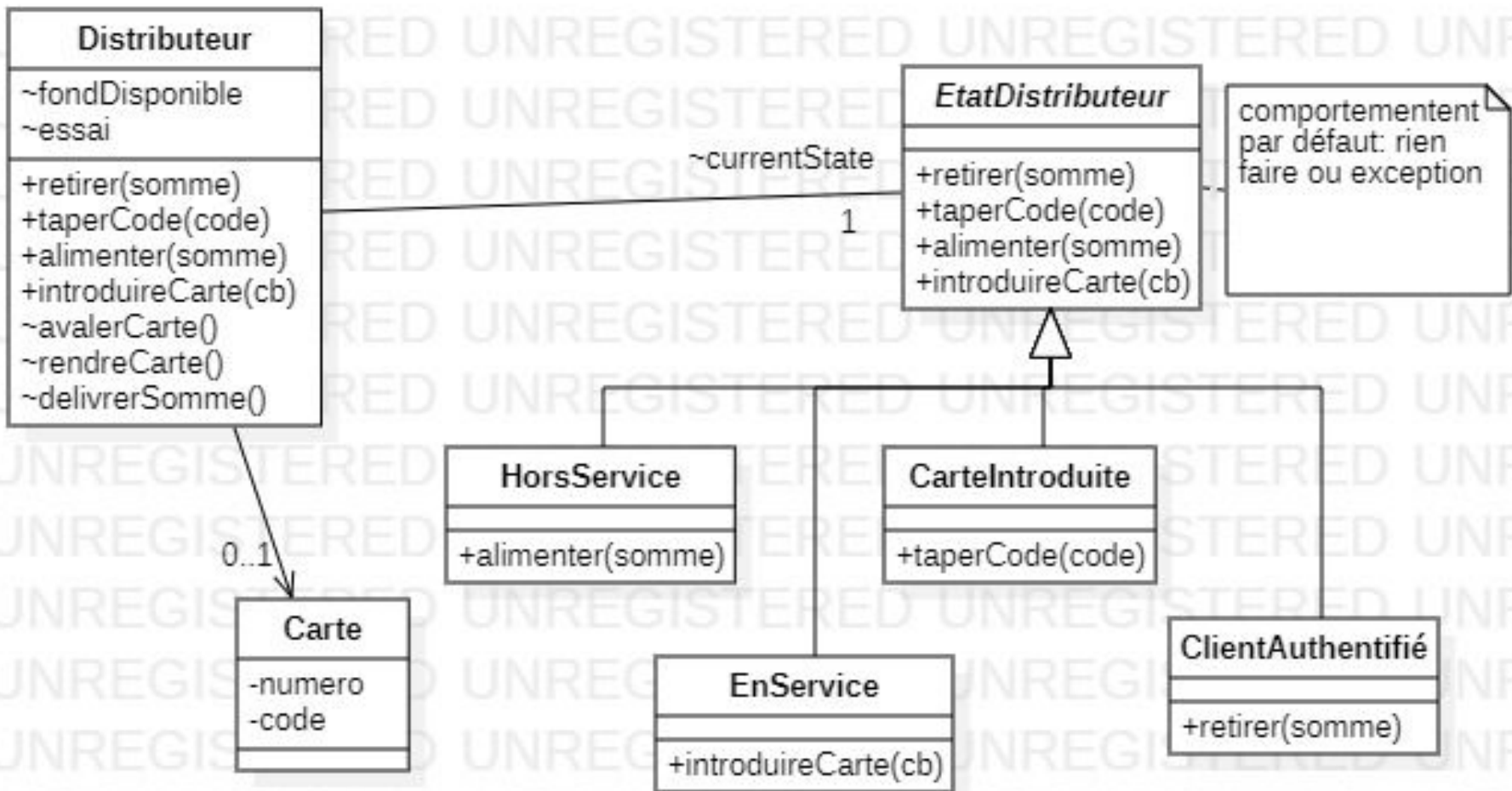
retirer

[sommeDemandée > fondDisponible]

[else]

[else] / fondDisponible -= sommeDemandé, délivrerSomme, rendreCarte





Bibliographie

- Design Patterns: Elements of Reusable Object-Oriented Software (1994)
The "Gang of Four": Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides
- Design patterns pour Java - Les 23 modèles de conception: description et solutions illustrées en UML 2 et Java (2009, 5ème édition 2022)
ENI Editions
Laurent Debrauwer
- Head First Design Patterns: Building Extensible and Maintainable Object-Oriented Software 2004, O'Reilly
Elisabeth Robson, Eric Freeman
Traduction française: Design Patterns - Tête la première
- Design Patterns in Java (2006)
Pearson Education Inc.
Steven John Metsker, William C. Wake
- Pattern-Oriented Software Architecture "POSA" vol 1 à 5 (1996-2007)

Autre classification des patterns du Gof

Interface: adapter, façade, composite, bridge

Responsability: singleton, observer, mediator, proxy, chain of responsibility, flyweight

Construction: builder, factory method, abstract factory, prototype, memento

Operation: template, state, strategy, command, interpreter

Extension: decorator, iterator

Autres patterns

Liste: POSA 1 à 5

- https://en.wikipedia.org/wiki/Pattern-Oriented_Software_Architecture
- https://en.wikipedia.org/wiki/Software_design_pattern

Quelques uns:

- MVC MVP, MVVP
- RAI
- Pool

Anti patterns

- Mauvaise pratiques

- God object
- too much YAGNI

- Code Smells

- Qualité de code

- SonarQube
- Linter
- Tests + Couverture
- Quality Gate

- Méthodes agiles