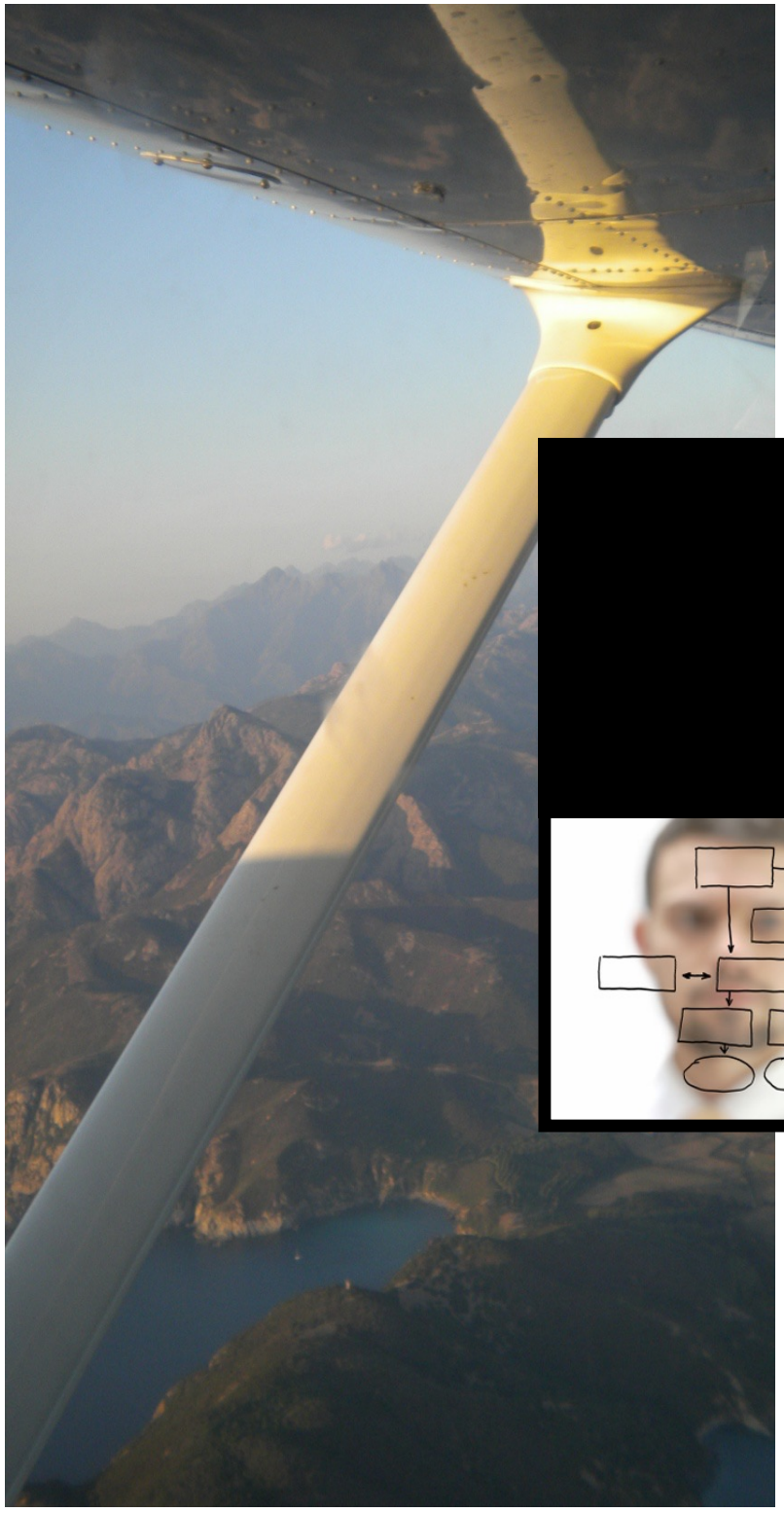
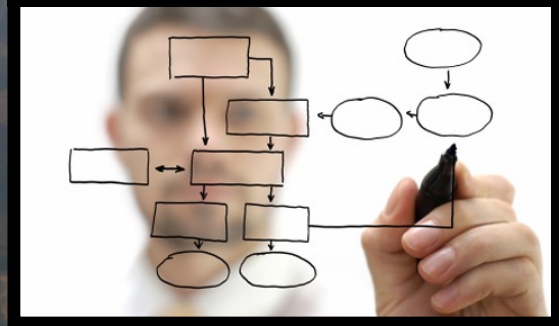


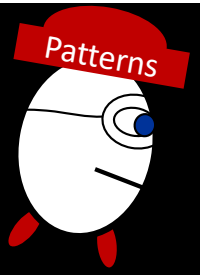
Université de Corse
2025-2026
MASTER Informatique DE-DFS
1ère année

PATTERNS

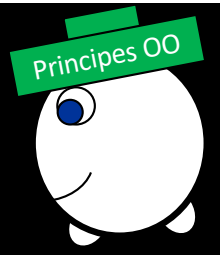
CH 2 - Patterns GOF



Evelyne VITTORI
Université de Corse
vittori_e@univ-corse.fr



Patterns de Conception
Plan du Cours



CH1 – Fondements de l'approche « Patterns »

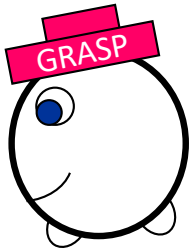
CH2 – Patterns Gof

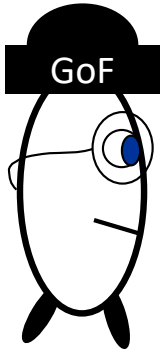
Introduction

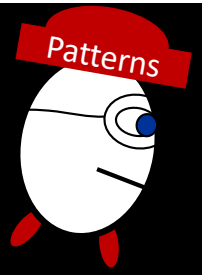
2.1 – Patterns créationnels

2.2 - Patterns structurels

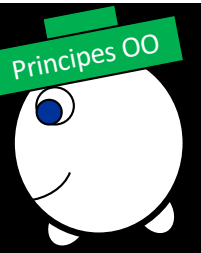
2.3 - Patterns comportementaux







Patterns Gof



Le "Gang of four“:

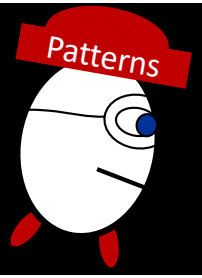
Erich Gamma,
Richard Helm,
John Vlissides,
Ralph Johnson

Un ouvrage de référence

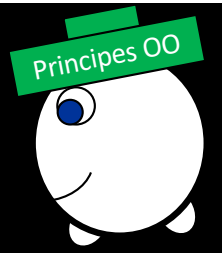
« Design patterns - Elements of reusable Object-Oriented Software »

Ed. Addison-Wesley 1995,1998

Version française
(Ed. Vuibert 1999)



Patterns Gof



Trois Catégories de pattern Gof

- **Patterns créationnels**

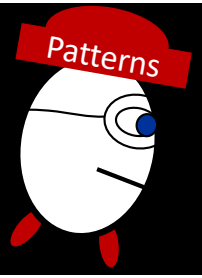
Organisation de classes permettant la création « déléguée » d'objets.

- **Patterns structurels**

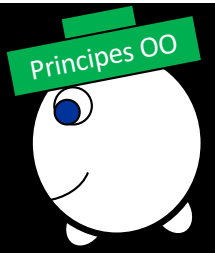
Modélisation de structures de données complexes.

- **Patterns comportementaux**

Organisation de classes axée sur la modélisation de comportements.



Patterns Gof



Le « tableau périodique » des 23 Patterns Gof

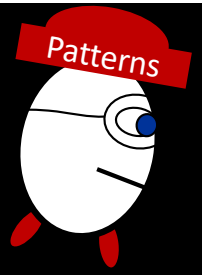
The Sacred Elements of the Faith

the holy origins									the holy structures
	FM Factory Method							A Adapter	
	PT Prototype	S Singleton	the holy behaviors			CR Chain of Responsibility	CP Composite	D Decorator	
	AF Abstract Factory	TM Template Method	CD Command	MD Mediator	O Observer	IN Interpreter	PX Proxy	FA Façade	
	BU Builder	SR Strategy	MM Memento	ST State	IT Iterator	V Visitor	FL Flyweight	BR Bridge	

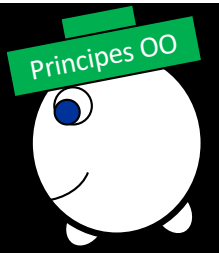
Source : http://home.earthlink.net/~huston2/images/periodic_table.png

Description d'un pattern Gof

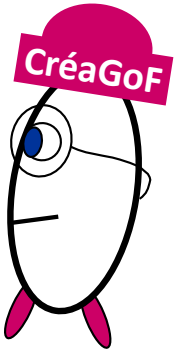
Nom	
Problème	Description du Problème résolu par le pattern (cas général)
Solution	Solution proposée (cas général)
Exemple(s)	1 – Illustration du problème 2 – Résolution du problème par application du pattern
Conséquences	Avantages et inconvénients du pattern Variantes et Conseils de mise en oeuvre



Patterns Gof



1 – Introduction



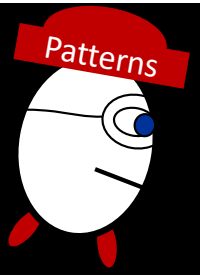
2 – Patterns créationnels

- Fabrique Abstraite (*Abstract Factory*)
- Fabrication (*Factory*)
- Monteur (*Builder*)
- Singleton

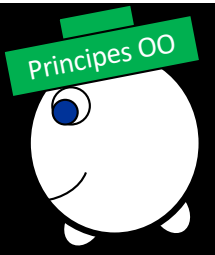
- Prototype

3 - Patterns structurels

4 - Patterns comportementaux

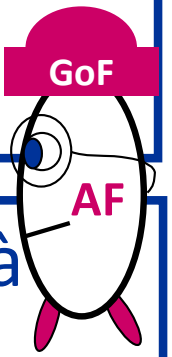


Fabrique abstraite «Abstract Factory»



Problème

Comment définir une **interface commune** pour créer des ensembles d'objets apparentés (**familles d'objets**)?



Exemple

Familles = ensemble d'accessoires relatifs à une tenue de football

Objets composants de chaque famille =

- Short
- Maillot

Famille TenueDomicile

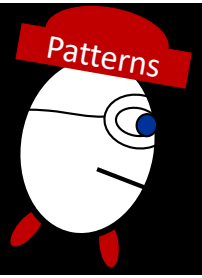
ShortNoir

MaillotBleu

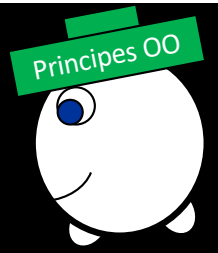
Famille TenueExtérieur

ShortRouge

MaillotBlanc

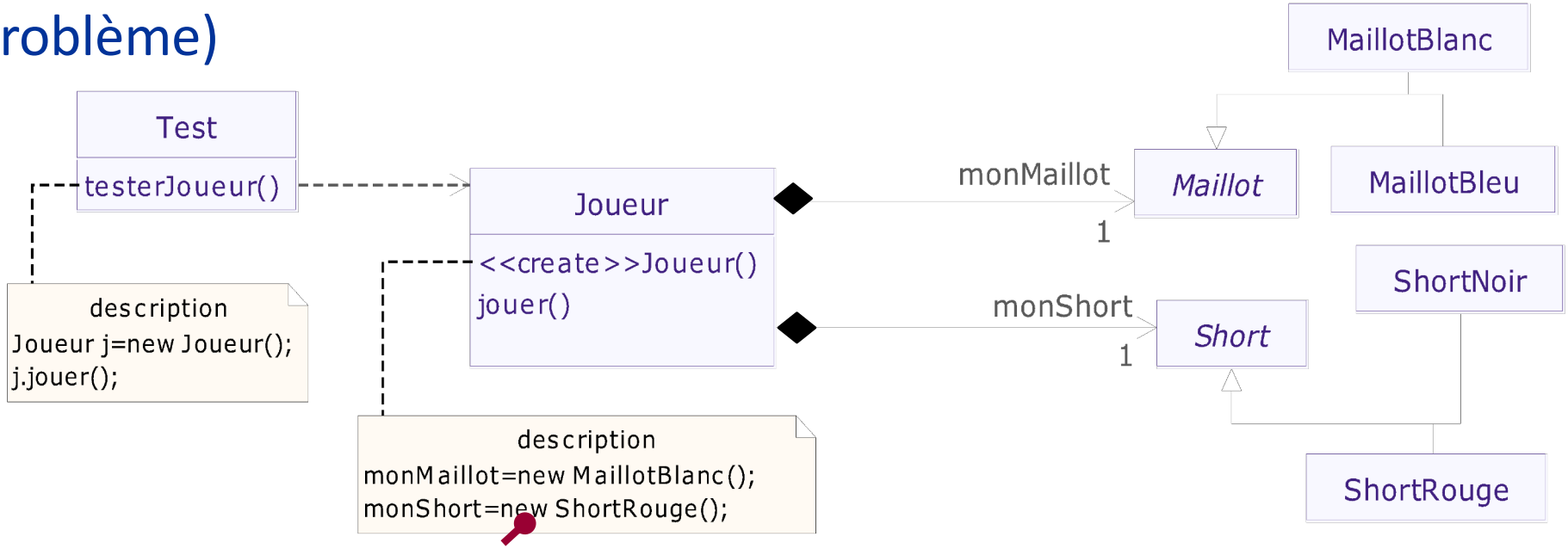


Abstract Factory

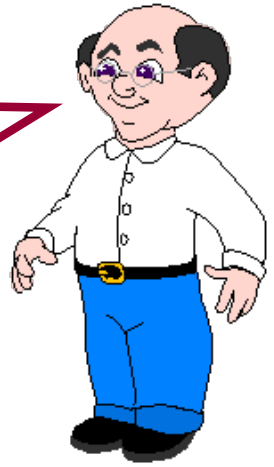


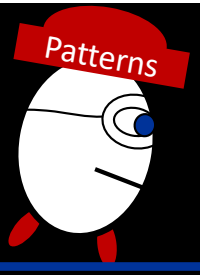
Exemple (Problème)

Types d'objets composants

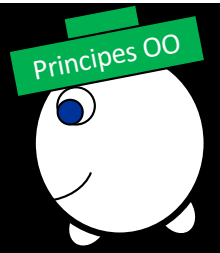


Non, Non, la création des accessoires n'est pas correcte!!
Mes joueurs ne pourront avoir que des tenues « extérieur »!
En plus, si la tenue extérieur change, je dois changer la classe Joueur!

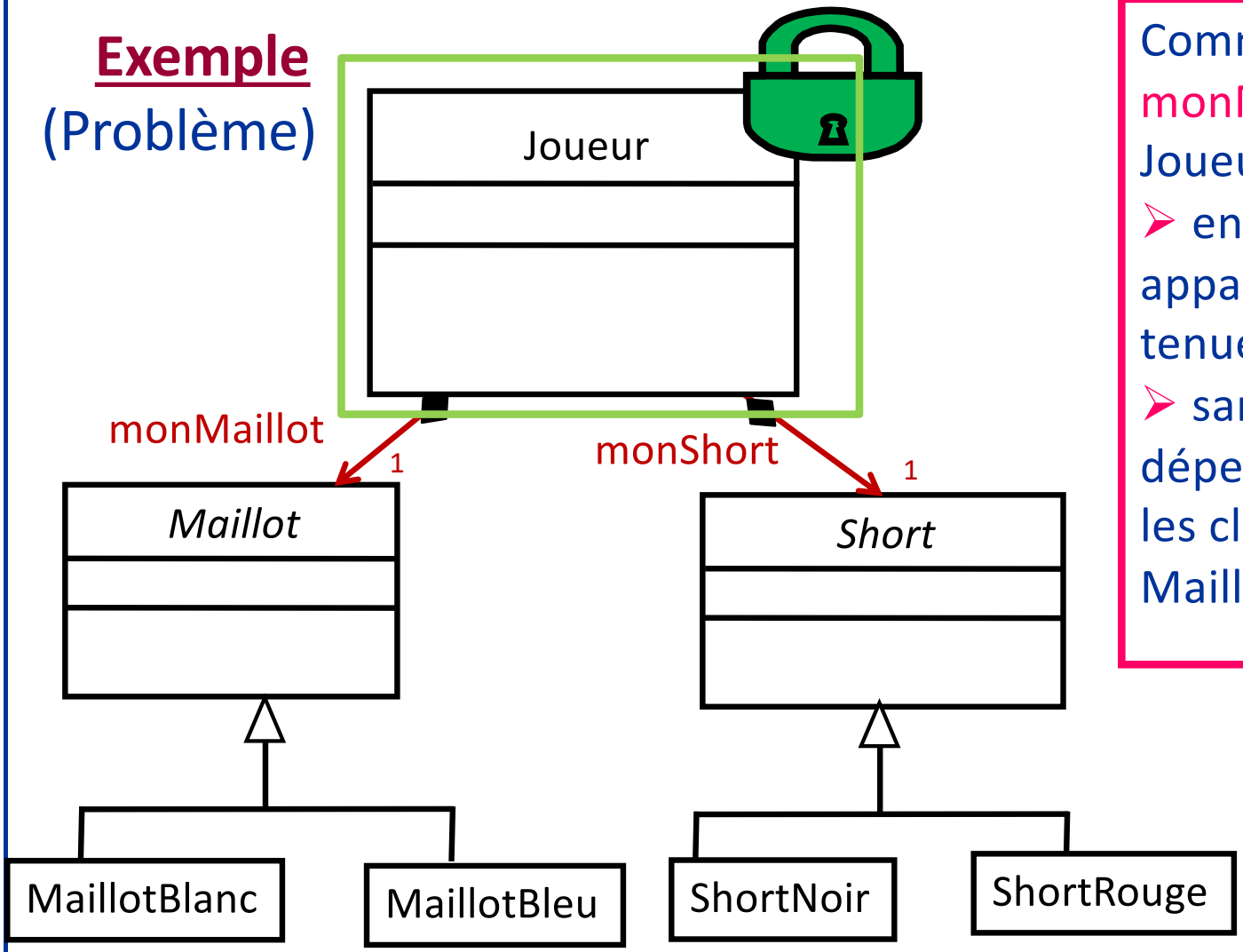




Abstract Factory

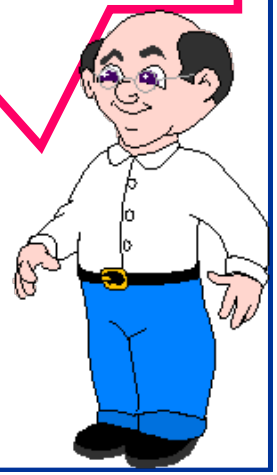


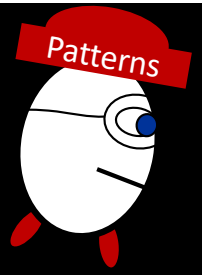
Exemple
(Problème)



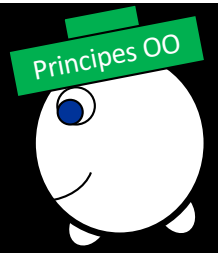
Comment instancier monMaillot et monShort dans Joueur??

- en étant certain qu'ils appartiennent à la même tenue (Domicile ou Extérieur).
- sans introduire une dépendance entre Joueur et les classes concrètes MaillotBlanc, ShortRouge, …





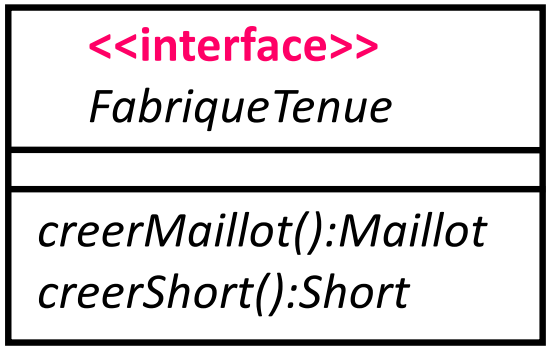
Abstract Factory



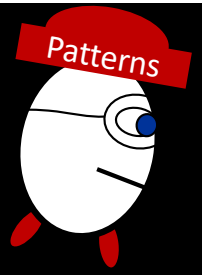
Exemple
(Problème)

Comment instancier une famille d'accessoires sans avoir à spécifier leurs classes concrètes?

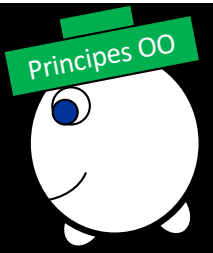
Exemple
(Solution)



1 - Définir une interface (ou une classe abstraite) (Fabrique Abstraite) comportant une méthode de création pour chaque objet composant à construire

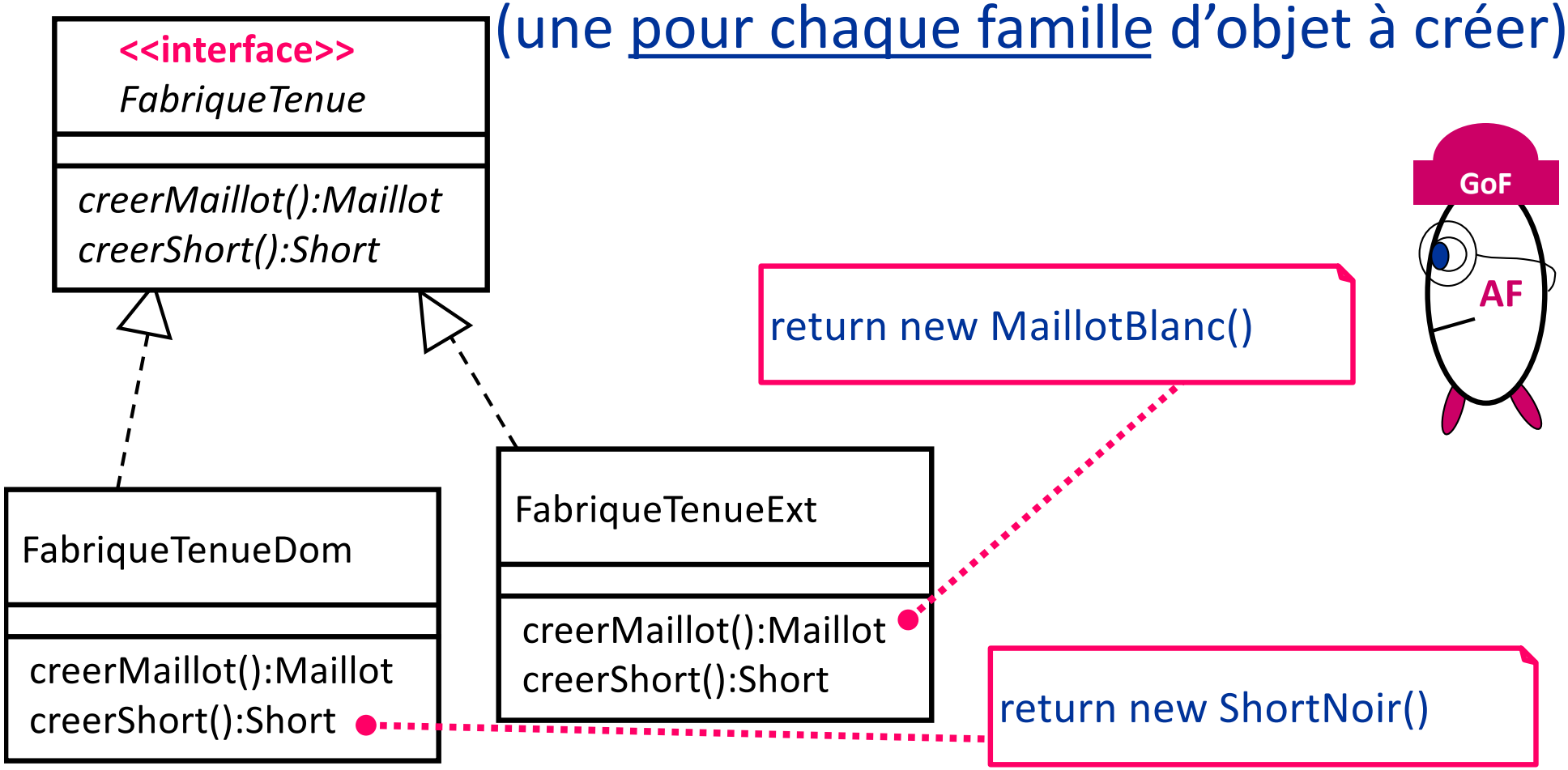


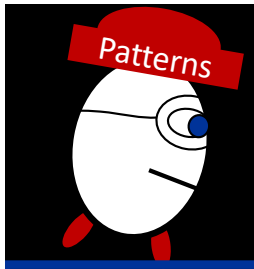
Abstract Factory



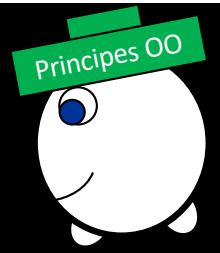
Exemple (Solution)

2 - Définir des classes Fabriques Concrètes (une pour chaque famille d'objet à créer)

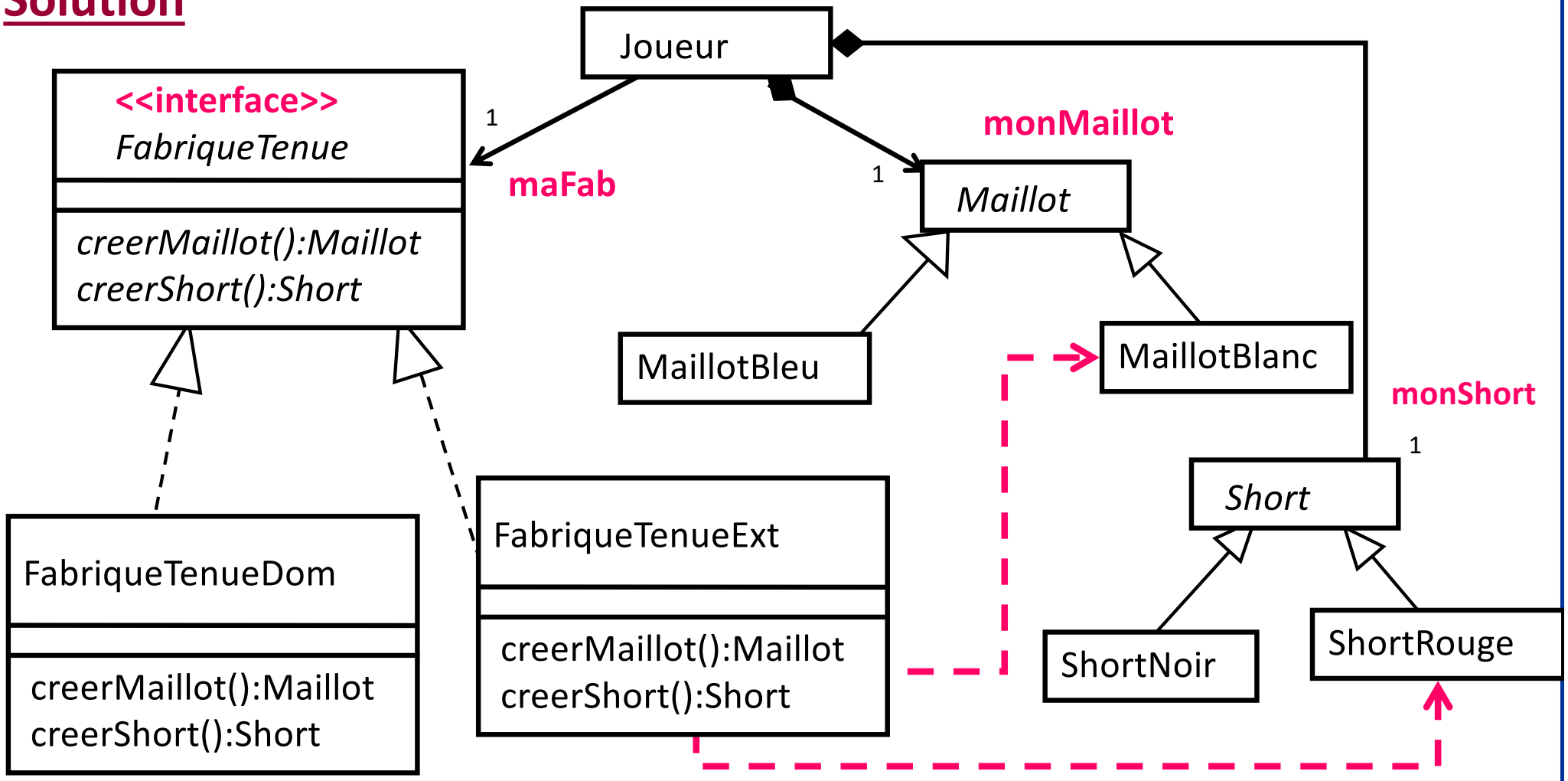


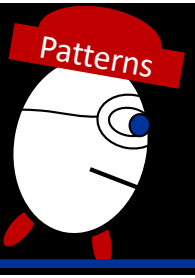


Abstract Factory

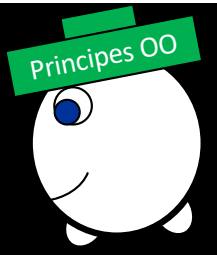


Solution

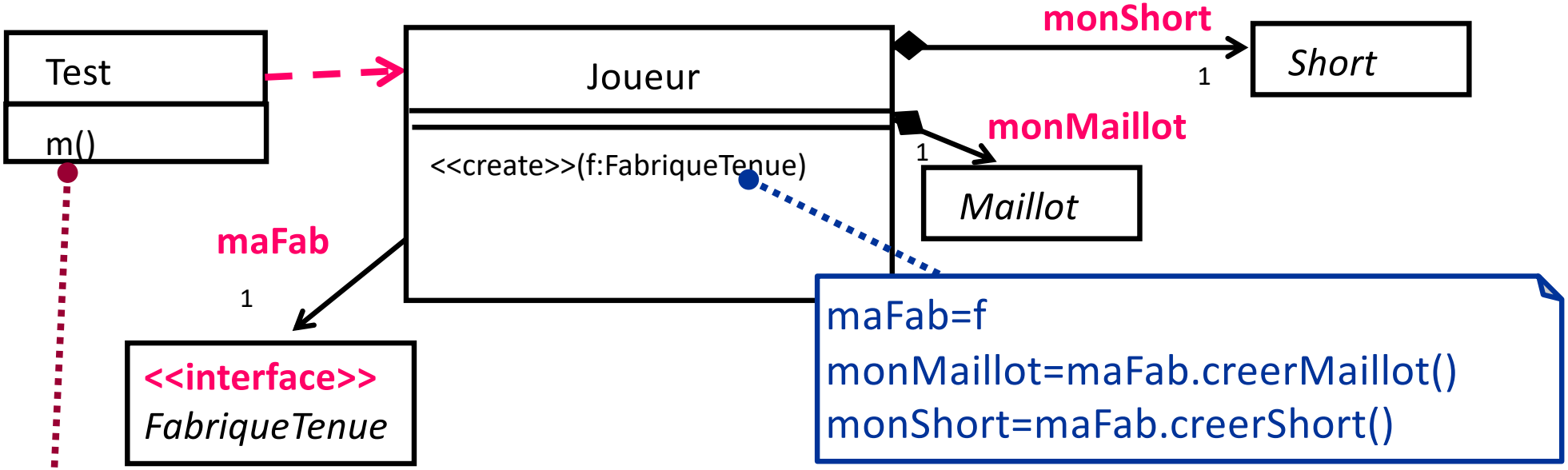




Abstract Factory

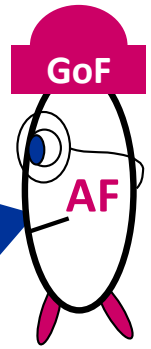


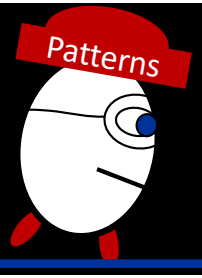
Solution (Mise en œuvre)



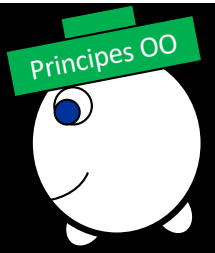

```
//Création d'un Joueur avec une tenue domicile
FabriqueTenue fab=new FabriqueTenueDom()
Joueur c=new Joueur(fab);
```

Joueur crée des accessoires de la même tenue sans dépendre des classes concrètes!!

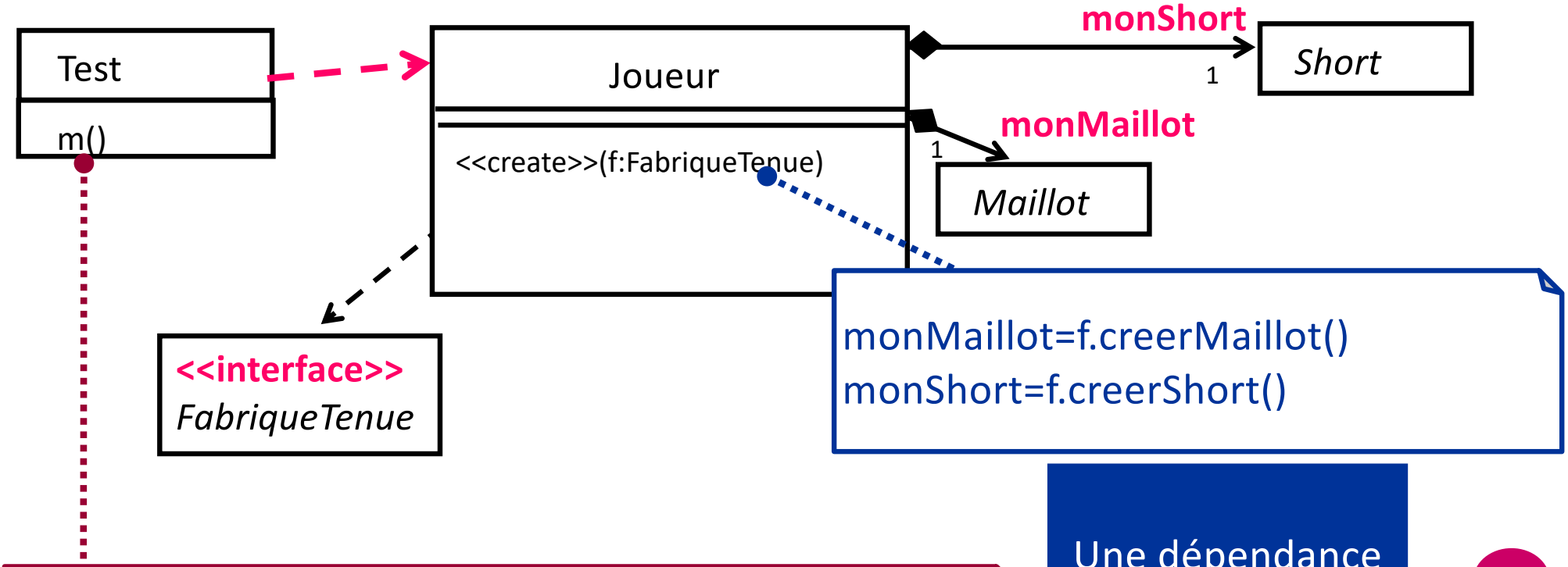




Abstract Factory

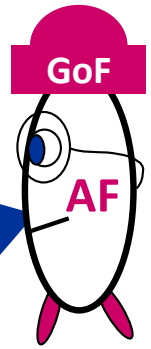


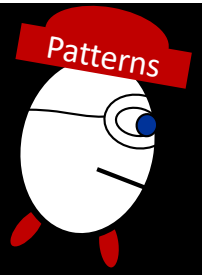
Autre Solution (Mise en œuvre)



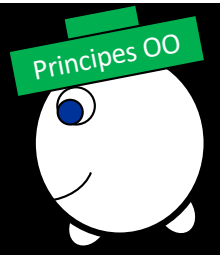

```
//Création d'un Joueur avec une tenue domicile
FabriqueTenue fab=new FabriqueTenueDom()
Joueur c=new Joueur(fab);
```

Une dépendance peut suffire si la fabrique n'est utilisée que dans le constructeur

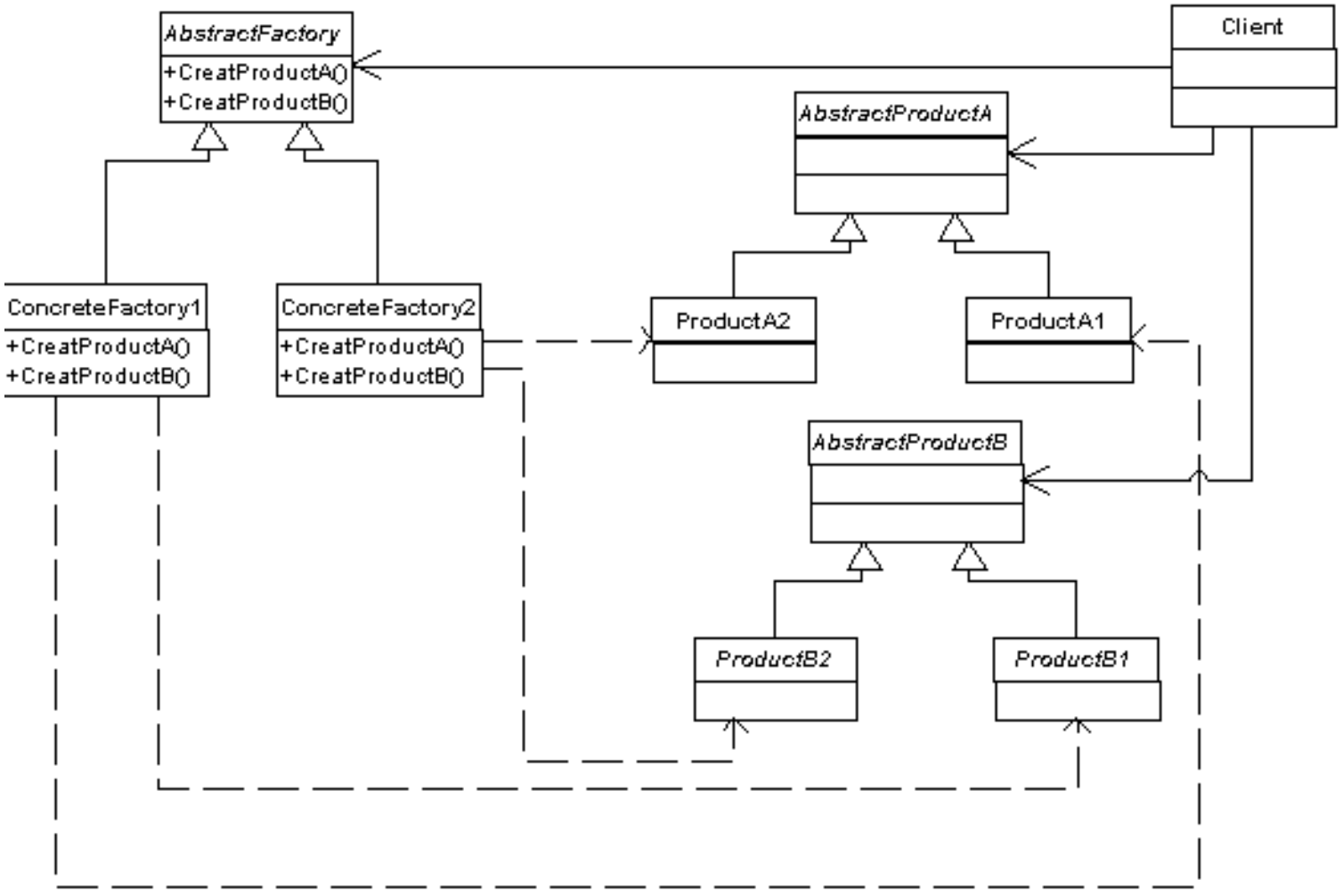


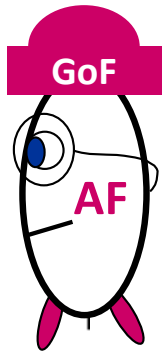


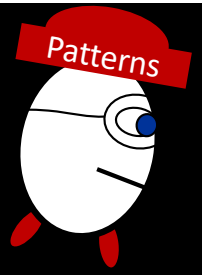
Abstract Factory



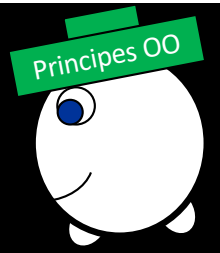
Solution (cas général)







Abstract Factory



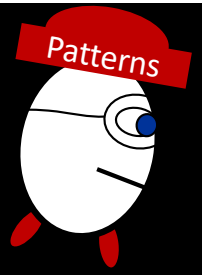
Conséquences

- L'ajout de nouveaux produits concrets et/ou de nouvelles familles d'objets peut se faire sans modifier le code existant:
  - nouvelles classes produits concrets
  - et/ou nouvelle classe fabrique concrète

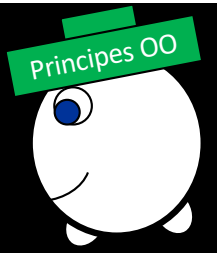
  (exemple: nouvelle tenue compétition)

- La prise en compte de nouveaux types d'objets composants (produits abstraits) s'avère fastidieuse:
  - ajout d'une méthode dans la Fabrique abstraite
  - ajout d'une méthode dans chaque Fabrique concrète

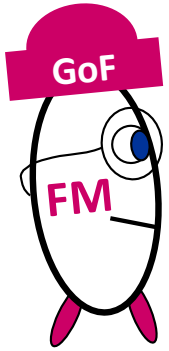
(exemple: prise en compte des chaussettes dans une tenue)



Fabrication «Factory Method»



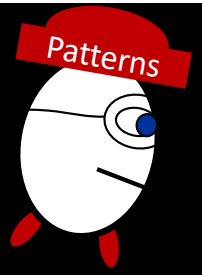
Problème



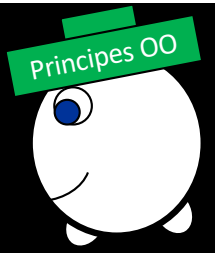
Comment une classe peut-elle créer un objet dont elle ne connaît pas le type concret?

Solution

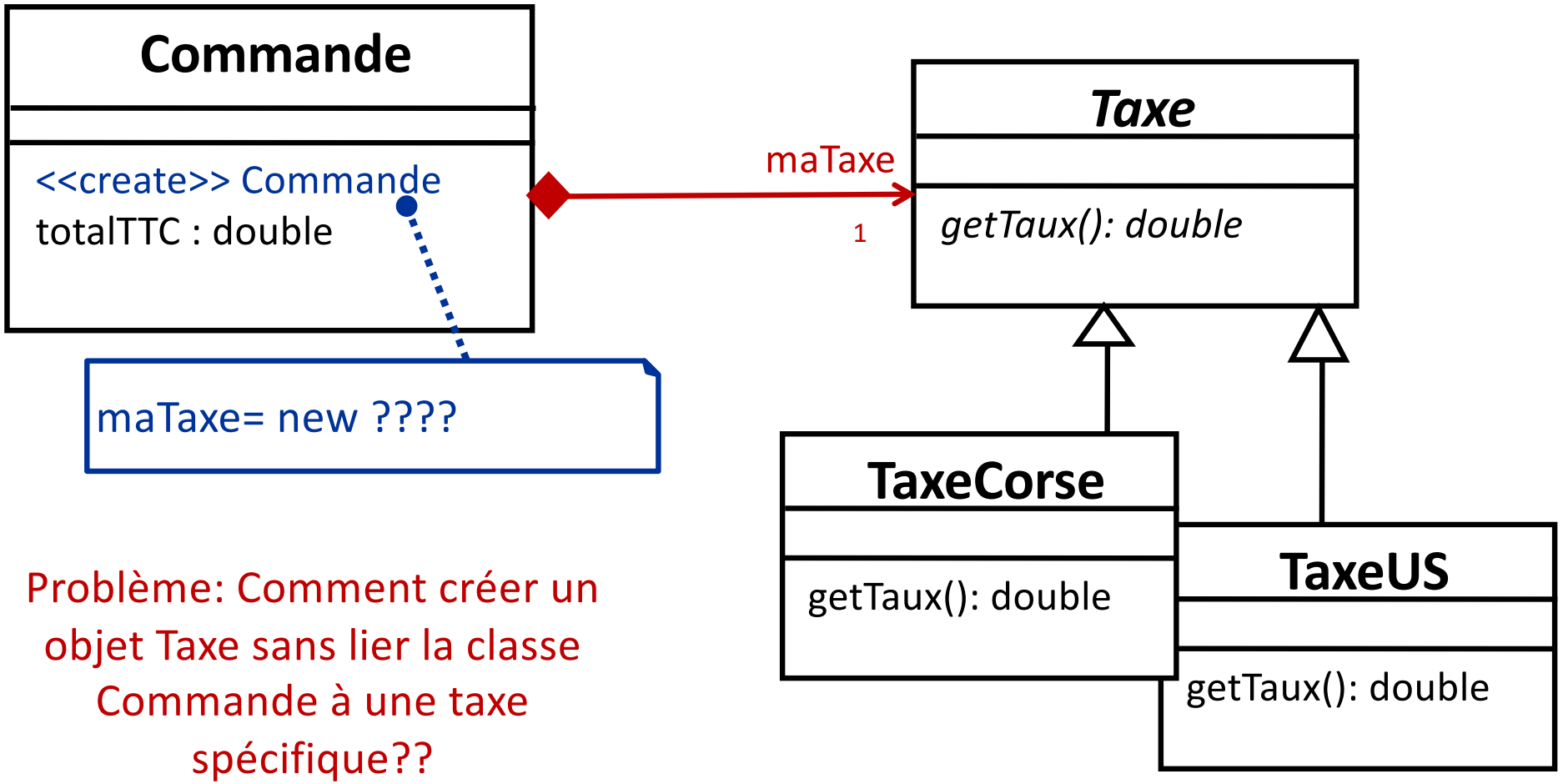
- Définir une classe abstraite contenant une **méthode abstraite de création** d'objets (instanciation)
- Définir des **sous-classes concrètes** implémentant réellement l'instanciation



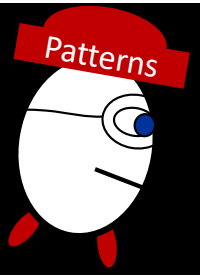
Factory Method



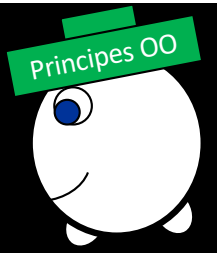
Exemple



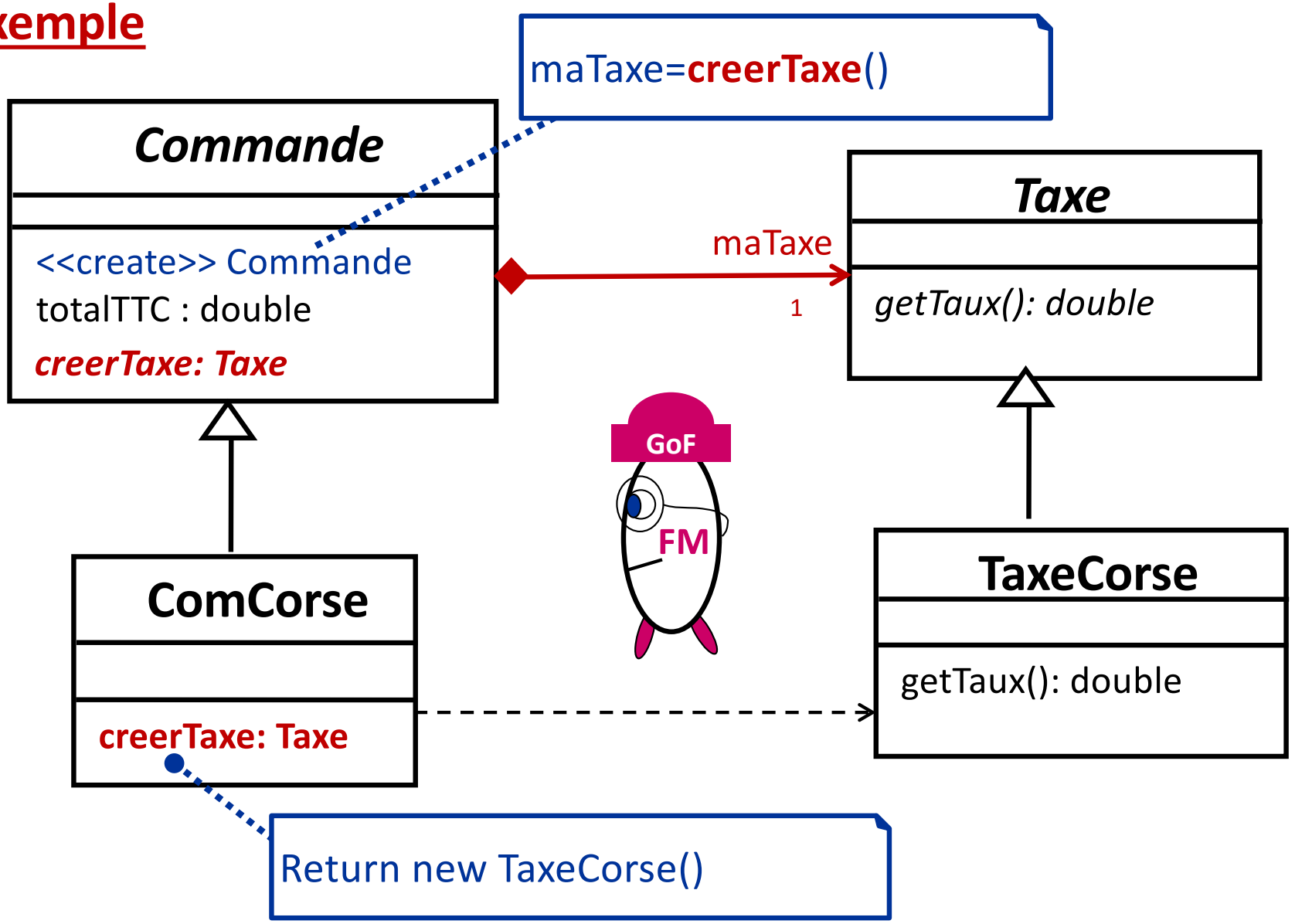
Problème: Comment créer un objet Taxe sans lier la classe Commande à une taxe spécifique??

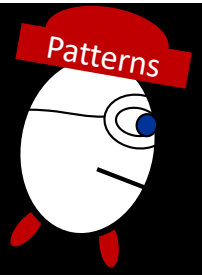


Factory Method

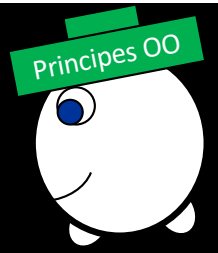


Exemple





Factory Method



Solution (cas général)

La « factory » method

Createur

#creerProduit(): Produit

+uneMethode()

Produit

....

Produit p=creerProduit()

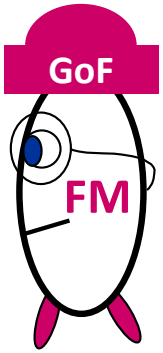
....

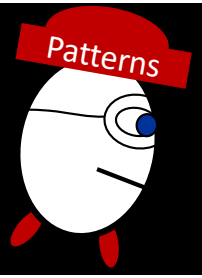
CreateurConcret

#creerProduit() : Produit

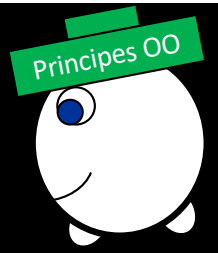
ProduitConcret

return new ProduitConcret()





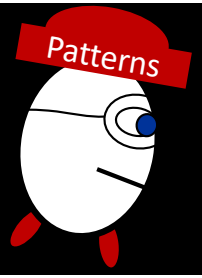
Factory Method



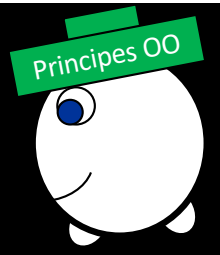
Conséquences

- Les clients auront besoin de créer une instance d'une sous-classe de la classe Créateur pour créer un produit concret.

- Les classes CreateurConcrets encapsulent tout le savoir-faire sur la manière d'instancier des produits concrets.

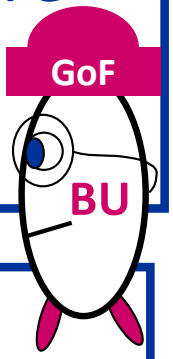


Monteur «Builder»



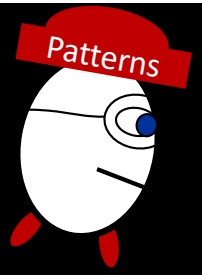
Problème

Comment définir les étapes du processus de création d'un objet complexe sans connaître sa structure interne?

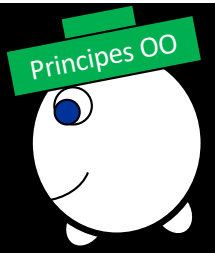


Exemple

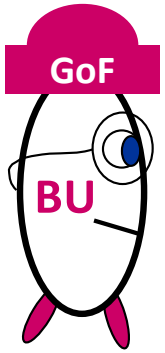
Un objet Agence souhaite pouvoir construire des Objets « Séjours » caractérisés par une liste d'hotels et d'excursions.



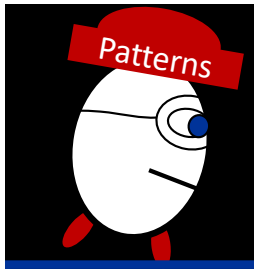
Builder



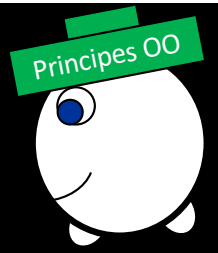
Solution



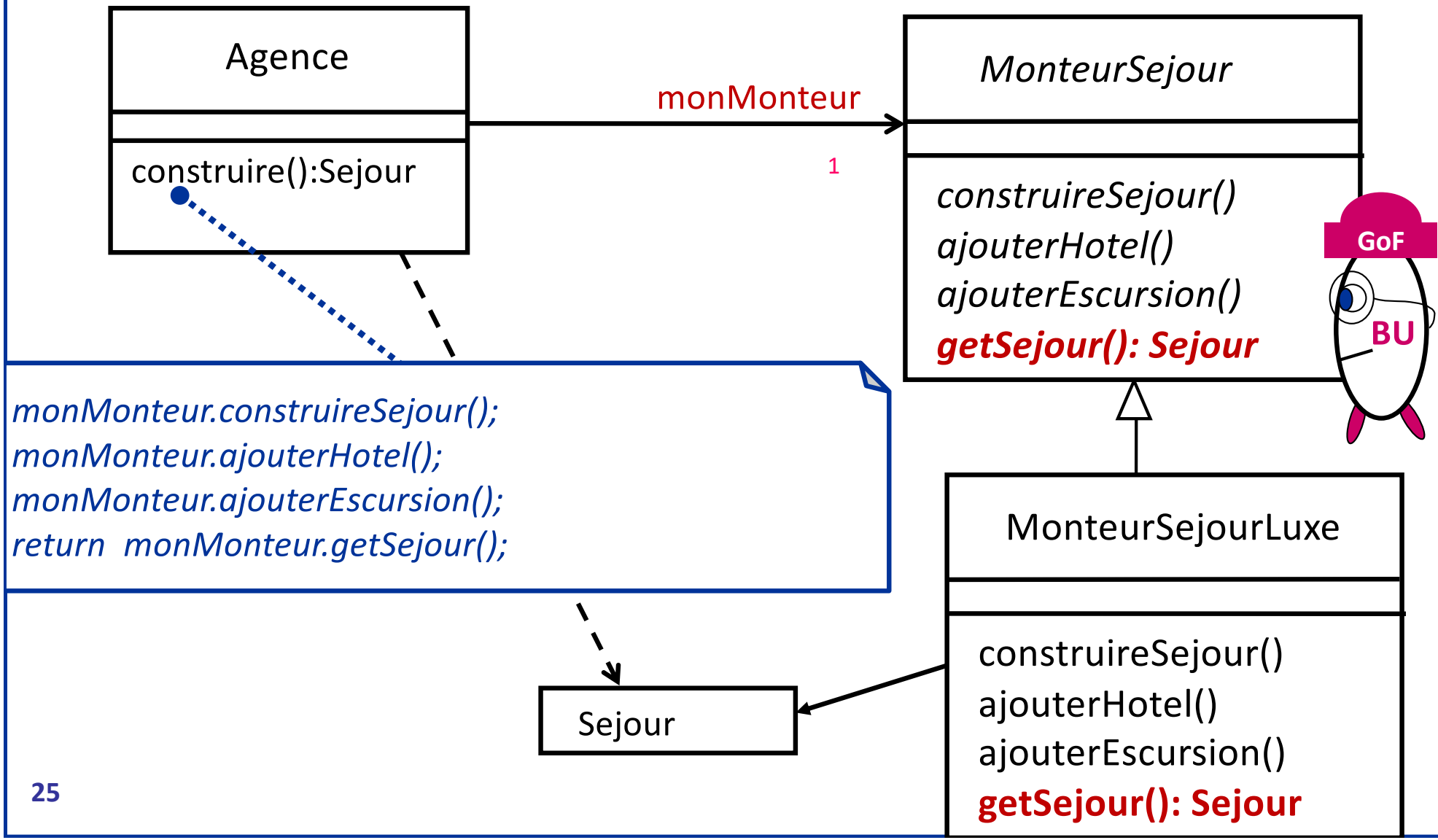
- Séparer le processus de construction de la représentation
- Encapsuler le processus dans une méthode faisant appel à un objet Monteur chargé de la création progressive de l'objet complexe

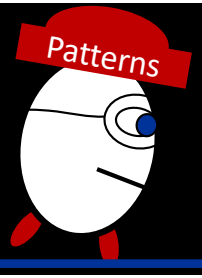


Builder

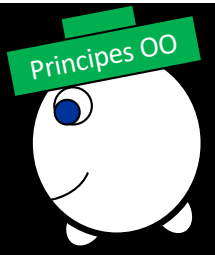


Exemple

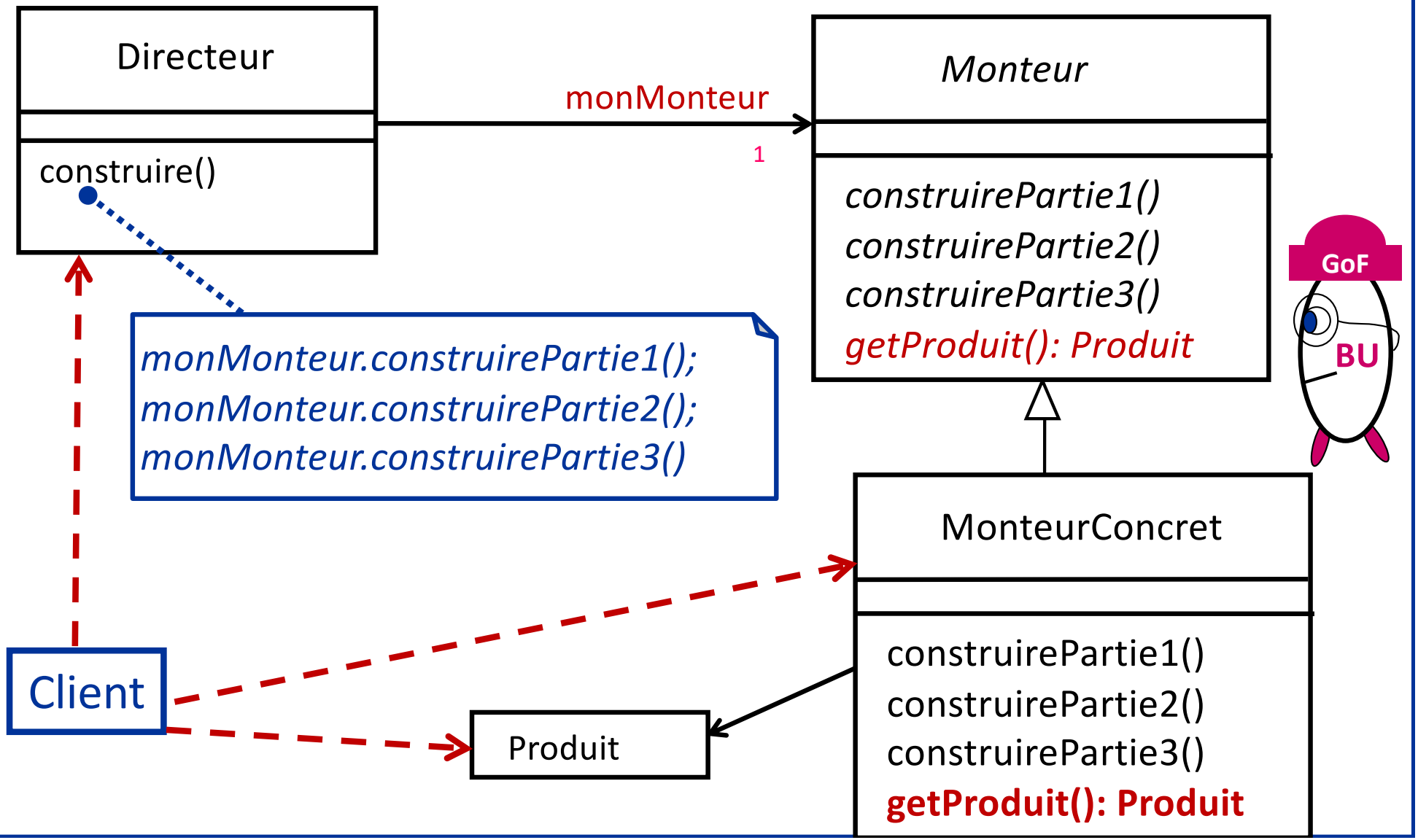


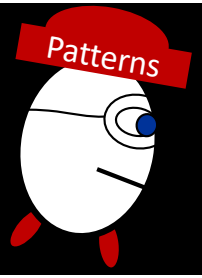


Builder

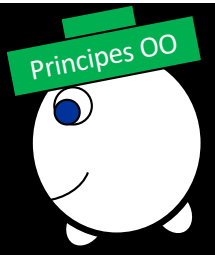


Solution (cas général)

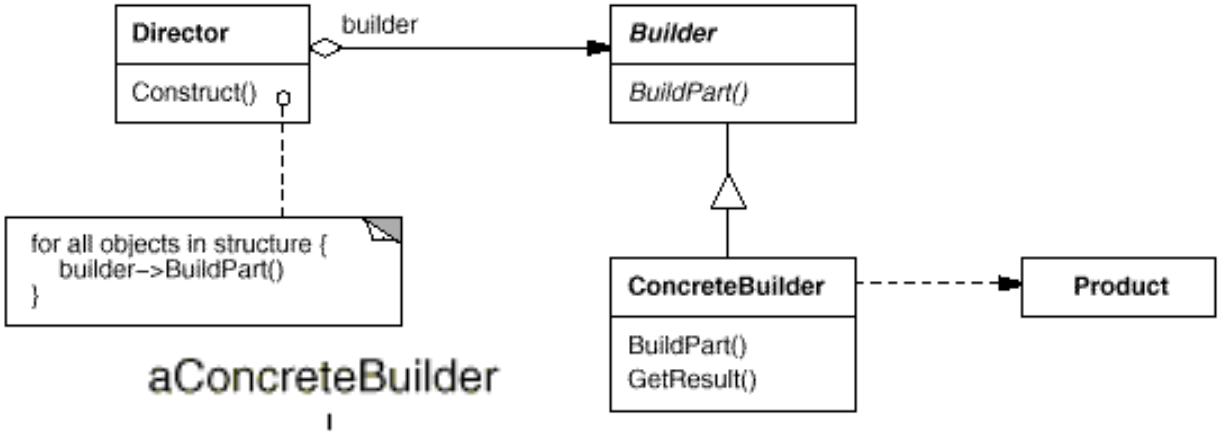


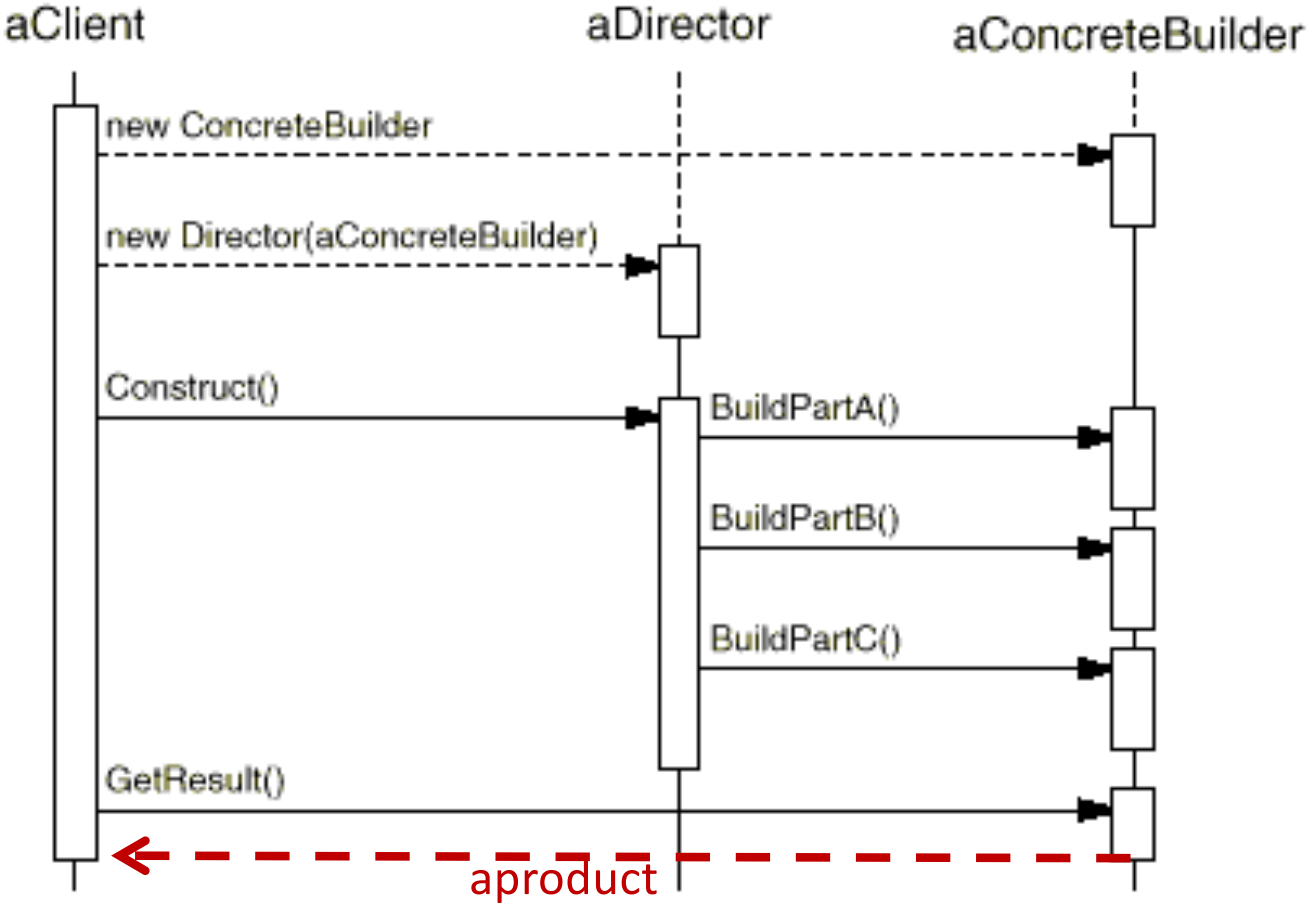


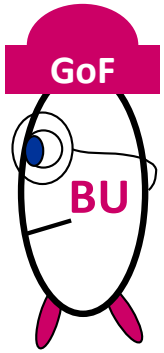
Builder

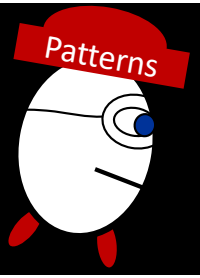


Solution (cas général)

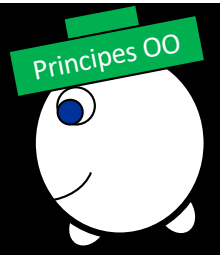




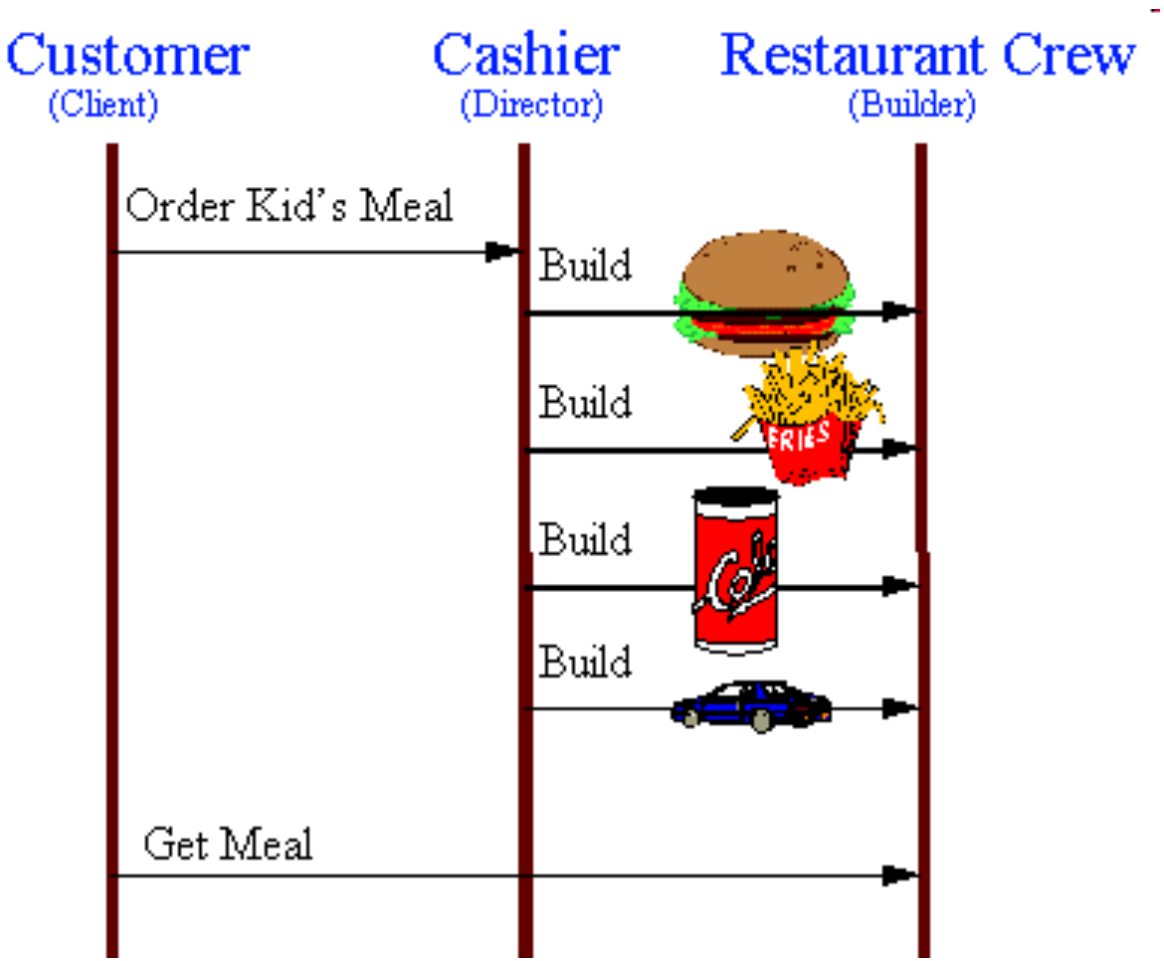


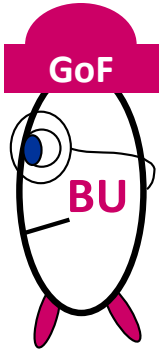


Builder

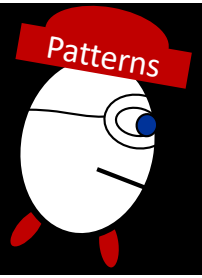


Exemple

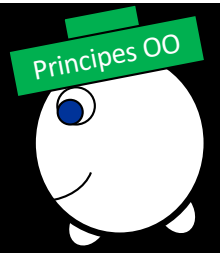




Source: "Non-software examples of software design patterns", *Object Magazine*, Jul 97, p54]

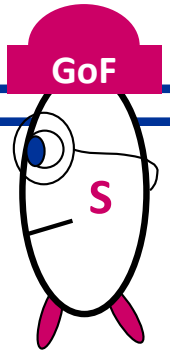


Singleton

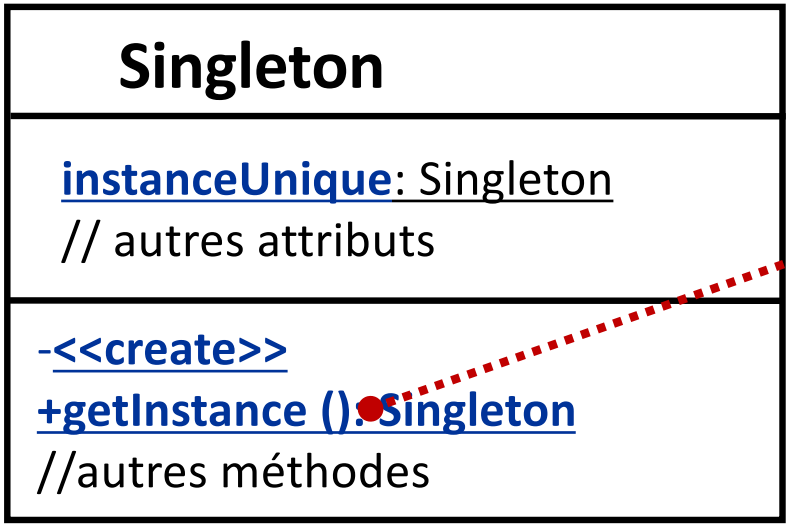


Problème

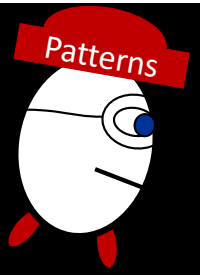
S'assurer qu'une classe ne possède **qu'une seule instance** et fournir les moyens d'y accéder.



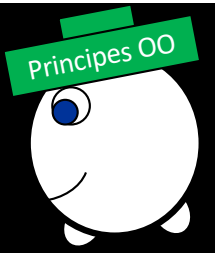
Solution




```
If  (instanceUnique==null ) {
    instanceUnique=new Singleton();
}
return instanceUnique;
```



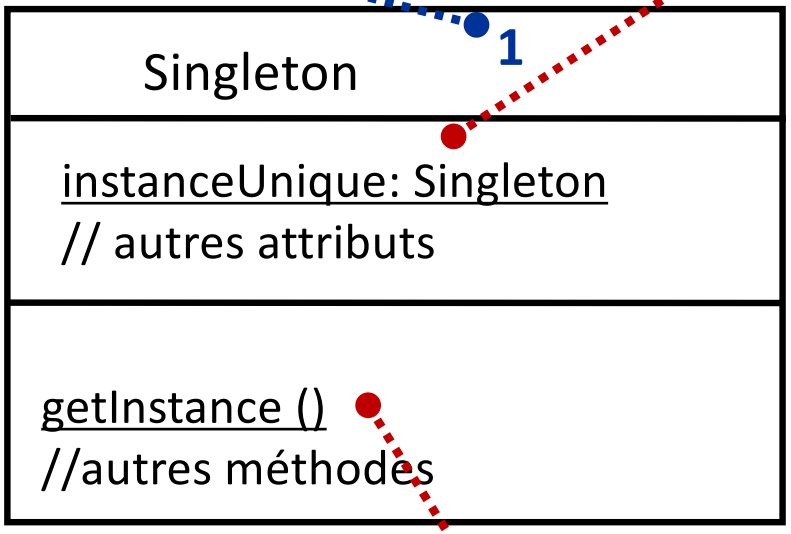
Singleton



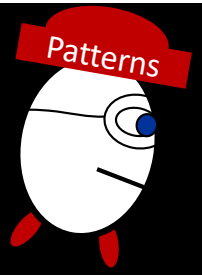
Solution (variante)

Indique que la classe est un singleton

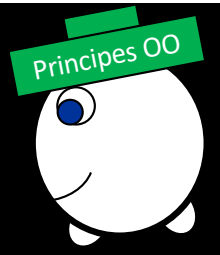
//valeur par défaut
Singleton instanceUnique=new Singleton()



Return instanceUnique;



Singleton

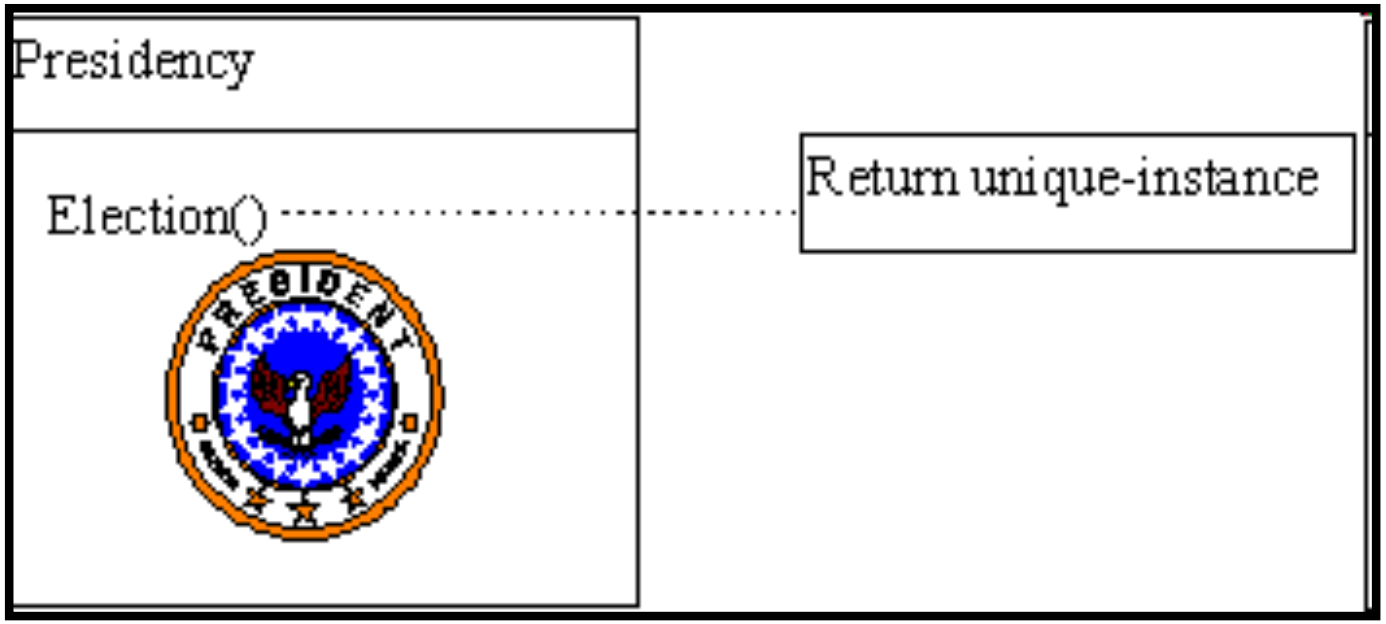


Conséquences

- Le constructeur de la classe Singleton est «protected ».
- Une fabrique concrete (cf. pattern Abstract Factory) est souvent implémentée par une classe Singleton.

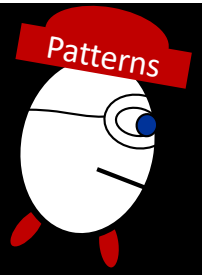
Singleton

Exemple

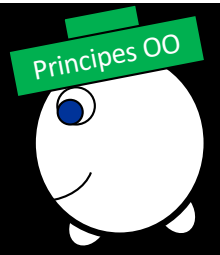


Source: "Non-software examples of software design patterns", *Object Magazine*, Jul 97

Retour Plan



Patterns Créationnels



Résumé

FACTORY METHOD	Les sous-classes décident quelles sont les instances à créer
ABSTRACT FACTORY	Permet à un client de créer des familles d'objets sans spécifier leurs classes concrètes
SINGLETON	Garantit qu'une instance et une seule ne peut être créée
BUILDER	Encapsule les étapes de création d'un objet complexe