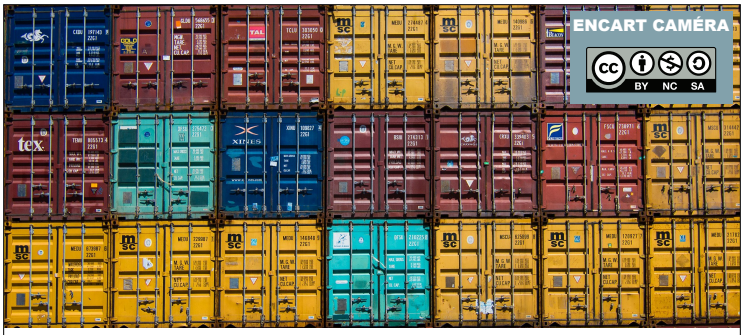



# Assembler des objets complexe : La relation de composition

**Sébastien Mosser** - INF5153
Chapitre 4 - Capsule 1
Automne 2020

ace





## Théoriser la relation de composition ?

ENCART CAMÉRA
BY NC SA

- On créé des **systèmes complexe** par **décomposition**
  - *Le classique "Diviser pour mieux régner"*
- En orienté-objet, la **relation de composition est duale** :
  - *Le fait qu'**un tout** soit **composé de parties***
  - *Le fait qu'**un objet délègue à un autre objet** un traitement*
- **Maintenir une bonne décomposition** est compliqué
  - Problème du "*path of least resistance*" lors d'une évolution.
  - *On fait apparaitre des "**Classes Dieu**" très facilement.*



## Publicité Outrancière

ENCART CAMÉRA
BY NC SA

ace

*A*bstract
*C*omposition
*E*ngine

Scan me

https://ace-design.github.io

## Cas #1 : Un tout est constitué de parties

ENCART CAMÉRA

- **Structurellement**, un ***élément*** fait partie d'un ***agrégat***
  - *Les cartes constituent la pile*
    - *La classe* **`CardPile`** *est un agrégat d'instances de* **`Card`**
  - *Une voiture contient 4 pneus*

- Dans ce cas, **la composition est intrinsèque à la situation**
  - c.à.d. "*les objets reflète la réalité*"

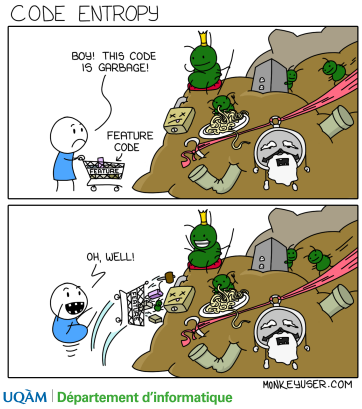


## Cas #2 : Délégation à un autre objet

ENCART CAMÉRA

- La composition est "**conceptuelle**" plus que structurelle
  - *Une partie de poker contient un croupier*
- On utilise ce type de composition pour **casser la complexité d'un objet** qui aurait tendance à centraliser trop de choses
  - Permet de **maintenir le principe de responsabilité unique**.
  - Notion de "*fournisseur de service*" pour la classe
- Il est **facile/dangereux** de rajouter des choses dans une classe :
  - *Pas besoin de réfléchir à l'encapsulation, on voit tout.*
  - **La délégation permet de diminuer l'entropie** dans le code.






ENCART CAMÉRA

***Que coûterait l'ajout d'une nouvelle fonctionnalité dans le code "en l'état" ?***

# Représentation de la Composition

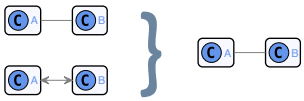


- On parle de **composition** dès qu'*un objet détient une référence vers un autre objet*.
- Le langage UML définit **trois (3) types de relations** pour ce cas :
  - L'**association** (flèche simple, *pas vraiment une composition*)
  - L'**agrégation** (diamant blanc ◇, *souvent utilisée pour le cas de délégation*)
  - La **composition** (diamant noir ◆, *composition structurelle forte*)
- Chaque type de relation à une **sémantique forte**
  - **Elles ne sont pas interchangeable (dans le détail) !**

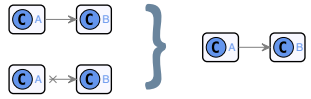



# Bonnes pratiques UML





*référence bidirectionnelle*



*référence unidirectionnelle*

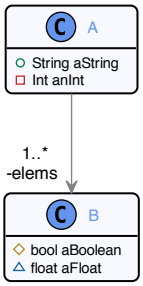


# Rappel : Le sens de lecture est inversé !



```
class A {
  public String aString;
  private int anInt;
  private Set<B> elems;
}

class B {
  protected boolean aBool
  float aFloat
}
```

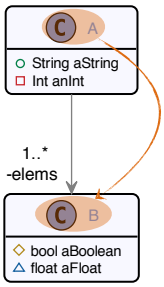


## Références ≠ Attributs



```
class A {
  public String aString;
  private int anInt;
  private Set<B> elems;
}

class B {
  protected boolean aBool
  float aFloat
}
```





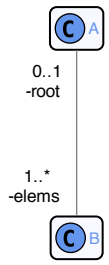

## Composition et Multiplicité



```
class A {
  private Set<B> elems;
}

class B {
  private A root;
}
```

***Attention, des infos du modèle de conception peuvent être "invisible" dans la structure du code.***



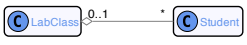

- **Soyez explicite** dans vos modèles si des **points sont importants** pour le code :
  - **`root`** : *null ? Optionnel ?*
  - **`elems`**: *ordonné ? Non-ordonné ? Taille min ?*



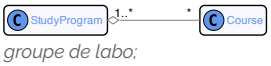
## Agrégation (◇) versus Composition (◆)



- **L'agrégation** représente une **composition "faible"**
  - "*La partie peut exister sans son tout*"
  - Définit un **graphe orienté acyclique** d'appartenance
    - **Cycles dans le modèle = problème de conception *(presque)***




- **Exemples** :




  - *Un étudiant fait partie (ou non) d'un seul groupe de labo;*
  - *Un cours peut faire partie de plusieurs programmes d'études.*

## Agrégation (◇) versus Composition (◆)

ENCART CAMÉRA

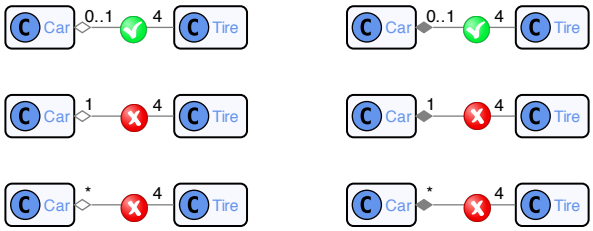
- **La composition** représente une **composition "forte"**
  - Si on **détruit l'agrégat**, on **détruit aussi l'élément**
  - La partie ne peut être **contenue** que par **un seul** agrégat
    - **Donc la multiplicité coté *agrégat* est toujours de 0 ou 1**


- **Exemple** :
  - *Si je détruit "INF5153", est-ce que je détruis aussi les étudiants ?*



## Agrégation (◇) versus Composition (◆)

ENCART CAMÉRA



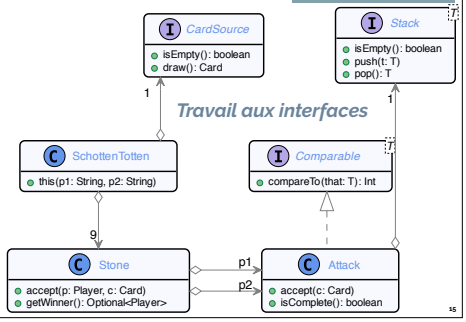

En pratique, **entre ◇ et ◆**, on peut souvent **approximer** ce qu'on veut dire en utilisant **toujours la même relation**.



## Problème de composition : Piles & Attaques

ENCART CAMÉRA

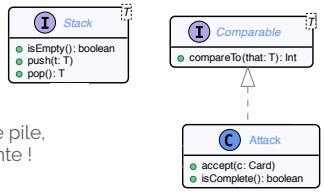
- Le jeu **SchottenTotten** contient neuf **Stones**
- Chaque **Stone** contient deux **Attacks** :
  - Celle de "p1"
  - Celle de "p2"
- Une **Attack** contient une **Stack** de cartes



## Une attaque contient une pile de carte ?

ENCART CAMÉRA

- Dans la conception précédente, on utilise une composition …
  - **Pourtant, une attaque, c'est une pile de carte non ?**
    - Donc ça devrait être une **réalisation** plutôt qu'une **composition** !
      - … Ça dépend !



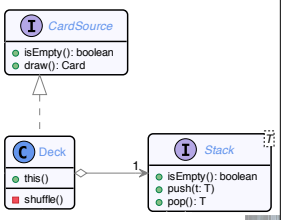

- **L'interface publique d'une attaque n'est pas celle d'une pile de carte**
  - Techniquement c'est (*peut-être*) une pile, mais la **logique d'affaire** est différente !



## Composition et Sous-Typage

ENCART CAMÉRA

- **Une relation de composition n'implique pas le sous-typage**
  - *Contrairement à la **généralisation** et la **réalisation***




- Dans cet exemple
  - Un `Deck` **est une** `CardSource` *(réalisation)*
  - Un `Deck` **contient** une `Stack` *(composition)*
    - ***Mais une `Stack` n'est pas une `CardSource`***



## Un 2e problème de composition : la multi-pile

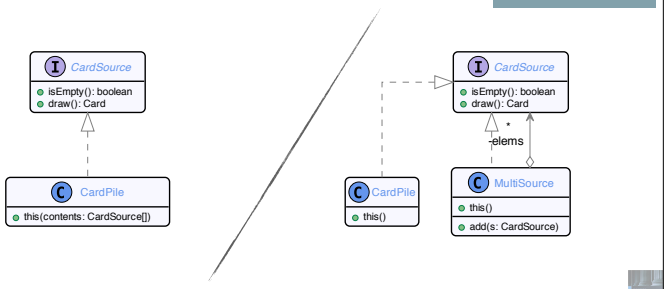
ENCART CAMÉRA

- **Comment jouer avec deux jeux de cartes ?**
  - *Voire "n" jeux de cartes ?*

- ***Deux conceptions opposées*** :
  - Fabriquer **une pile qui contient toutes les cartes** des autres
    - *Se pose le problème de la copie (voir chapitre précédent)*
  - Fabriquer **une pile qui est composée des autres piles**
    - *Une sorte de "pile composite" (analogie avec des LEGO©)*

## Copie versus Composition

ENCART CAMÉRA



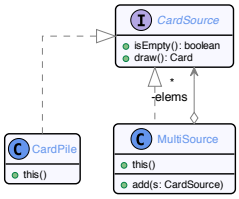

**Comment faire évoluer la version par copie ?**



## Analyse de l'approche par composition

ENCART CAMÉRA

- **Forces** :
  - ***MultiSource*** *réalise* ***CardSource***
  - *Couplage faible*
  - *On peut rajouter des sources dynamiquement*
  - *Pas de copie en mémoire*
- **Faiblesses** :
  - *Plus difficile à mettre en oeuvre*
  - *Pour ajouter une source, il faut savoir que c'est une* ***MultiSource***



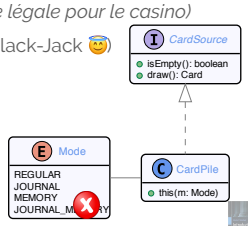

 *Cette forme est un "**patron de conception**" (le "composite"), on l'étudiera en détail au **chapitre 7***


## Dernier problème : Ajouter du comportement

ENCART CAMÉRA

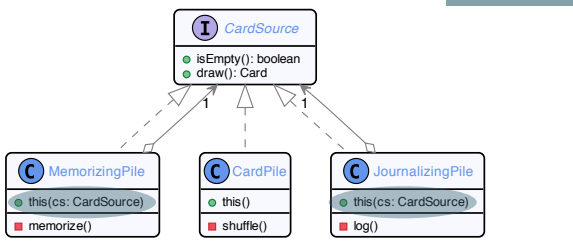
- On veut une **`CardSource`** qui puisse :
  - **Faire son travail** normalement;
  - **Journaliser les actions** faites dessus *(exigence légale pour le casino)*
  - **Mémoriser les cartes** tirées (pour gagner au Black-Jack 😇)
- **Proposition**
  - On rajoute un "**mode**" dans la source,
  - et on décide en fonction !

**Switch Statement**
**Explosion Combinatoire !**



## Proposition : Composer les comportements !



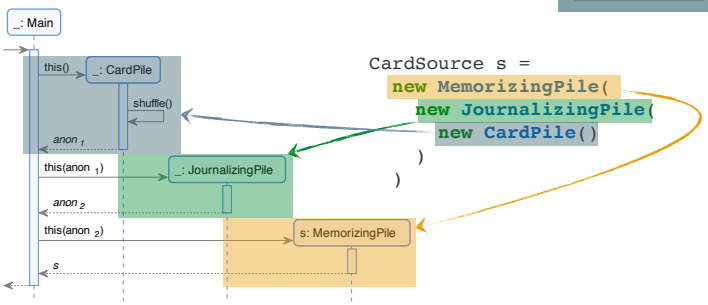



```
CardSource s =
  new MemorizingPile(new JournalizingPile(new CardPile()))
```



## Création des objets de gestion de la pile






```
CardSource s =
  new MemorizingPile(
    new JournalizingPile(
      new CardPile()
    )
  )
```

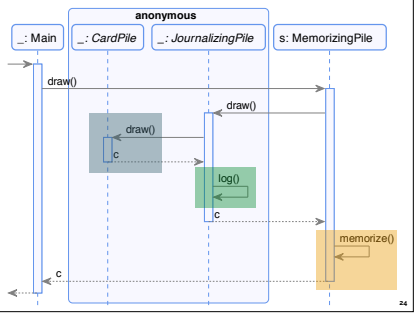


## A l'exécution, on exploite la délégation



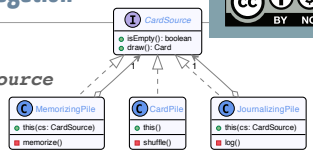
```
CardSource s =
  new MemorizingPile(
    new JournalizingPile(
      new CardPile()
    )
  )

s.draw();
```



# Analyse de l'approche par délégation





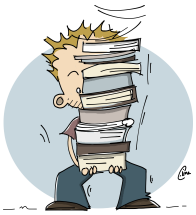

- **Forces** :
  - *Réalisation homogène de* **`CardSource`**
  - *Couplage faible*
  - *On peut rajouter des comportement dynamiquement pendant l'exécution*
  - *Responsabilité unique et ouvert-fermé en même temps*
- **Faiblesses** :
  - *Pleins de petits objets, dont l'orde de construction importe.*
  - *Problème de l'identité entre objets. Comment définir* **`equals`** *?*



UQÀM | Département d'informatique

FACULTÉ DES SCIENCES
Université du Québec à Montréal





https://mosser.github.io/

ace

https://ace-design.github.io/

***Abonne toi** à la chaine,*
*et met un pouce bleu !*