ENCART CAMÉRA

# General Responsibility Assignment Software Patterns (aka GRASP)

**Sébastien Mosser** - INF5153
Chapitre 5 - Capsule 2
Automne 2020

ace



UQÀM | Département d'informatique

*crédits photos: Pixabay*

---

ENCART CAMÉRA

# Ça va sans le dire ...

## ... mais c'est mieux quand on l'a dit !

UQÀM | Département d'informatique

---


ENCART CAMÉRA

Les exemples du Monopoly sont inspirés de diapositives crées par Sylvain Cherrier, Maitre de Conférences à l'université Paris-Est Marne-La-Vallée.

# Patron #1 : Spécialiste de l'Information

ENCART CAMÉRA

- **Situation** :
  - **A qui donner une responsabilité ?**
- **Proposition** :
  - Donner la responsabilité à la classe qui **connait** les informations permettant de **faire** cela.
- **Exemple** :
  - *Au Monopoly, on a des **joueurs** qui se déplacent sur des **cases** disposées sur un **plateau** de jeu.*
  - **A qui donner la responsabilité d'accéder à une case du Jeu ?**

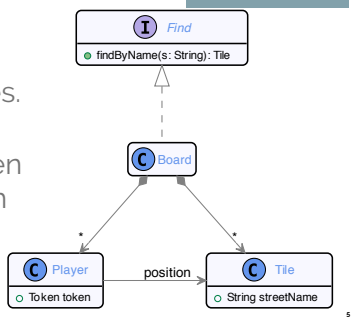


# Application au Monopoly

ENCART CAMÉRA

Le plateau **connait** les cases.

il est **le SPÉCIALISTE** pour en **faire la recherche** par nom






# Récapitulatif

ENCART CAMÉRA

- **Patron le plus utilisé** pour affecter les responsabilité
  - **C'est globalement du bon sens**
  - "*Ça va sans le dire, mais c'est toujours mieux quand on l'a dit*"
- **Principe de base** en conception orientée objet
  - ***L'encapsulation repose intrinsèquement sur ce patron***
- **Question à se poser** :
  - *Qui dispose de l'information nécessaire à la réalisation de cette tâche ?*

# Avantages & Inconvénients

ENCART CAMÉRA

- **Bénéfices** :
  - **Favorise** la création de classes **cohésives** et **encapsulées**
  - **Distribue** par essence **le comportement** à travers les objets
    - *Pas de gros BLOB* qui concentre tout le comportement entouré de classes étant uniquement des structures de données ne rendant aucun service.*
- **Limitation** :
  - *L'accomplissement d'une responsabilité nécessite souvent que l'**information nécéssaire soit partagée** entre différents objets.*



* Le blob est un "anti-patron", on en parle au chapitre 8



# Patron #2 : Créateur

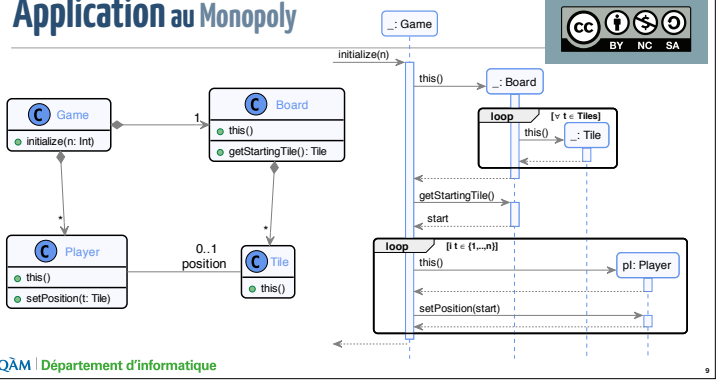
ENCART CAMÉRA

- **Situation** :
  - **Qui prend la responsabilité de créer une instance de classe ?**
- **Proposition** :
  - Affecter à la classe C la responsabilité de création des instances de C' si par exemple :
    - C est composée d'instances de C'
    - C à des données permettant d'initialiser les instances de C'
- **Exemple** :
  - **Qui crée les cases de jeu au Monopoly ?**



# Application au Monopoly

ENCART CAMÉRA



## Récapitulatif

ENCART CAMÉRA

- Attribution de la **responsabilité de créer** les objets
  - *On passe notre temps à créer des objets*
- **Question à se poser** :
  - *Quelle classe est la plus à même de créer cet objet ?*
  - *Quelle classe contient cet objet ?*
- *Le problème est souvent lié à l'"expertise en information"*



## Avantages & Inconvénients

ENCART CAMÉRA

- **Bénéfices** :
  - **Faible couplage** des objets *(pas de new à tout bout de champ)*
  - Moins de **dépendances**, meilleure **réutilisation**
  - Permet de faire des **optimisations de création**
    - *On peut mettre en place du recyclage d'objets (p.-ex. avec un bassin)*
- **Limitation** :
  - Pas toujours évident quand les **objets sont partagés**
  - Problème des **liens bidirectionnels** entre objets



## Patron #3 : Faible Couplage

ENCART CAMÉRA

- **Situation** :
  - **Minimiser les dépendances entre les objets et réduire l'impact des changements (p.-ex. lors des évolutions)**
- **Proposition** :
  - Lors de l'ajout d'une dépendance entre deux objets, regarder s'il n'existe pas une autre solution qui réduirait le couplage
- **Exemple** :
  - **A qui donner la responsabilité de déplacer le joueur sur le plateau?**

## Exemples de couplage



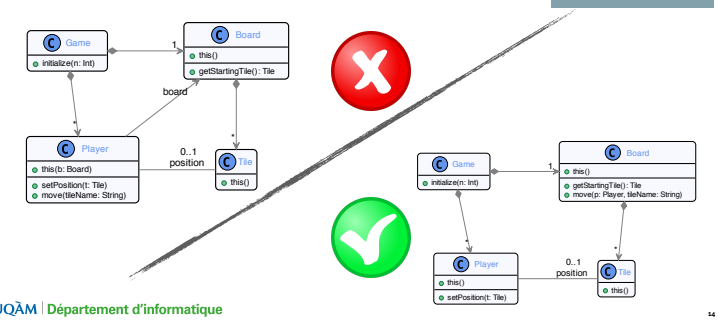
- Un type **X est couplé à** un type **Y** quand :
  - X a un **attribut de type Y** (composition)
  - X a une **méthode qui utilise Y** (dépendance)
  - X est un **sous-type de Y** (réalisation)
    - X est une **sous-classe de Y** (généralisation)

- En gros, **dès qu'il y a un trait** dans le diagramme UML, **c'est couplé**
  - *Ou si vos diagrammes de séquence **concentrent les envois de messages** vers d'autres objets*



## Application au Monopoly








## Récapitulatif



- Un **couplage fort** force à **changer tout ou une partie des classes** couplée lors d'une évolution
  - *Il n'y a pas de mesure absolue de quand un couplage est trop fort : "Ça dépend" !*
- Le "***path of least resistance***" d'évolution va souvent attaquer le couplage faible en en **rajoutant inutilement**
- Un couplage fort n'est pas forcément un problème si les **éléments couplés sont stables** (p.-ex. `java.util`)
- **Question à se poser** :
  - *Est-ce que cet objet a VRAIMENT besoin de connaître celui-ci ?*

# Avantages & Inconvénients

ENCART CAMÉRA

- **Bénéfices** :
  - **Faible couplage** des objets *(pas de new à tout bout de champ)*
  - Moins de **dépendances**, meilleure **réutilisation**
  - Permet de faire des **optimisations de création**
    - *On peut mettre en place du recyclage d'objets (p.-ex. avec un bassin)*
- **Limitation** :
  - Pas toujours évident quand les **objets sont partagés**
  - Problème des **liens bidirectionnels** entre objets



# Patron #4 : Contrôleur

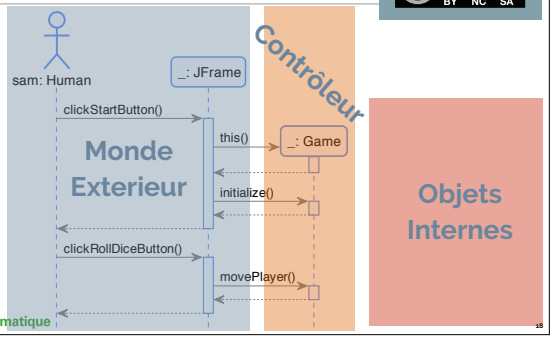
ENCART CAMÉRA

- **Situation** :
  - **Comment coordonner les messages provenant de l'extérieur (p.-ex. de l'IHM) sans coupler le modèle objet à l'extérieur ?**
- **Proposition** :
  - Inventer un objet qui va servir de zone tampon entre le système et l'application objet
- **Exemple** :
  - **Au Monopoly, qui coordonne le jeu ?**



# Application au Monopoly

ENCART CAMÉRA



## Récapitulatif

ENCART CAMÉRA

- Si vous avez entendu parler des architectures suivants le paradigme **Modèle - Vue - Contrôleur (MVC*)**
  - Le **Contrôleur** fait le lien entre le ***Modèle*** et la ***Vue***
- Permet de **maintenir le système objet isolé** du monde extérieur
  - *Éviter le "code marionnette" qui dépend de choses incontrôlées*
- **Question à se poser** :
  - *Est-ce que j'ai besoin d'un contrôleur dans le système ?*
  - *Est-ce qu'il est inhérent à la logique d'affaire (p.-ex. la game de monopoly), ou relié à un cas d'utilisation (p.-ex. xxxHandler)*



** Les architectures actuelles utilise plutôt MVVM, une évolution de MVC*



## Avantages & Inconvénients

ENCART CAMÉRA

- **Bénéfices** :
  - Maintien de **l'isolation** et améliore la **réutilisabilité**
  - Permet de **contrôler l'accès** au système objet
  - Un contrôleur peut **déléguer à un autre** (composition)
- **Limitation** :
  - On a tendance à **abuser de ce patron** et à créer des classes Dieu qui contrôlent des structures de données sans aucun comportement.



## Patron #5 : Forte Cohésion

ENCART CAMÉRA

- **Situation** :
  - **Comment s'assurer que les objets restent compréhensibles et faciles à gérer tout en contribuant à un faible couplage ?**
- **Proposition** :
  - Attribuer les responsabilités de telle sorte que la cohésion soit forte
  - Appliquer ce filtre pour choisir entre plusieurs solutions
- **Exemple** :
  - **Comment afficher la grille de Monopoly, respecter les règles et connaitre l'état du jeu ?**

# Cohésion ?

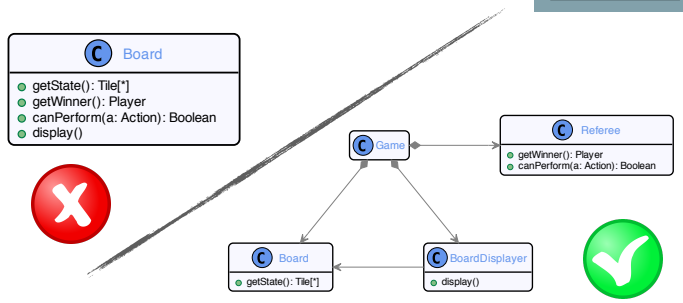
ENCART CAMÉRA

- La cohésion est une **mesure de l'étroitesse des liens** et de la spécialisation des responsabilités d'un élément
- Une classe qui a des **responsabilités étroitement liées** les unes aux autres et qui **n'effectue pas un travail gigantesque** est **fortement cohésive**
- *"Un objet bien conçu renferme une valeur lorsqu'il possède une telle quantité d'affordances que les personnes qui l'utilisent peuvent l'employer à des fins que le concepteur n'avait même pas imaginées"* - Donald Norman, 1994
  - Pas de définition "formelle". **Ça dépend**.



# Application au Monopoly

ENCART CAMÉRA






# Récapitulatif

ENCART CAMÉRA

- Une classe de forte cohésion a un petit nombre de méthodes, avec des fonctionnalités hautement liées entre elles, et ne fait pas trop de travail
- **Question à se poser** :
  - *Est-ce que je peux décrire ma classe avec une seule phrase ?*

# Avantages & Inconvénients

ENCART CAMÉRA

- **Bénéfices** :
  - **Maintenance** et **évolutivité** améliorées
  - Meilleur potentiel de **réutilisation**
  - Pas de "**code spaghetti**"
  - Meilleure **lisibilité**
- **Limitation** :
  - **Difficile à maintenir sur la durée**
    - path of least resistance : "*je met ce code ici, ça va plus vite*"



# Patron #6 : Polymorphisme

ENCART CAMÉRA

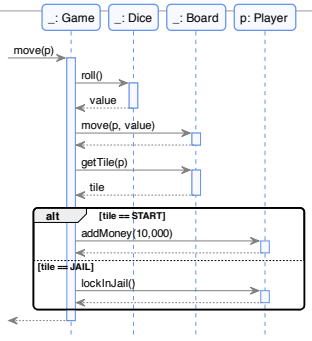
- **Situation** :
  - **Comment gérer des alternatives structurelles ?**
  - **Comment créer des composants "puzzle"**
- **Proposition** :
  - Affecter la responsabilité aux types et en proposer plusieurs réalisations alternatives qui peuvent être interchangées.
- **Exemple** :
  - **Comment gérer les cases de jeu différentes au Monopoly ?**



# Application au Monopoly

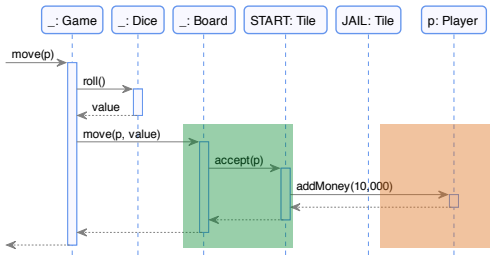
ENCART CAMÉRA

***Switch Statement !***



# Application au Monopoly

ENCART CAMÉRA



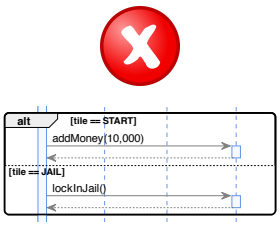

*Envoi de message !*

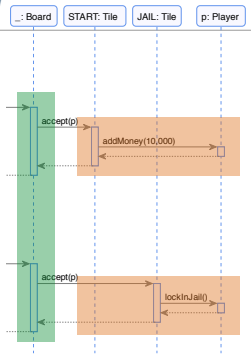
*Comportement différent selon le récepteur*



# Application au Monopoly

ENCART CAMÉRA



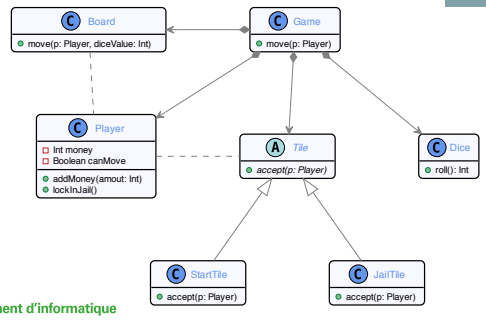




*Uniformité !*



# Diagramme de classe associé

ENCART CAMÉRA



## Récapitulatif

ENCART CAMÉRA

- Le polymorphisme repose sur le **mécanisme de sous-typage**
- On **évite d'écrire de gros blocs conditionnels**
- On laisse un **objet décider** du comportement

- **Question à se poser** :
  - *Existe-t'il plusieurs manières de réaliser ce service ?*
  - *Est-ce que ça dépend du type ou de l'instance ?*



## Avantages & Inconvénients

ENCART CAMÉRA

- **Bénéfices** :
  - Met en oeuvre le **principe Ouvert/Fermé**
  - Les **points d'extensions** sont clairement **identifiés**
  - On peut **introduire de nouvelles implémentations** facilement
    - *sans affecter les consommateurs existants*
- **Limitations** :
  - Signatures polymorphes parfois **difficile à identifier**
  - Certains langages **limitent la hiérarchie** d'héritage



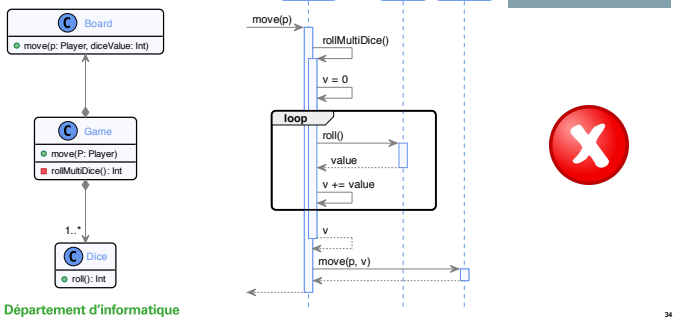
## Patron #7 : Fabrication Pure

ENCART CAMÉRA

- **Situation** :
  - **Comment faire quand les objets du monde réel (objets du domaine d'affaire) ne sont pas utilisables en faible couplage et forte cohésion ?**
- **Proposition** :
  - Affecter un ensemble de responsabilités fortement cohésives dans une classe créée artificiellement pour l'occasion
- **Exemple** :
  - **Comment lancer plusieurs dés pour déplacer son pion ?**

# Application au Monopoly

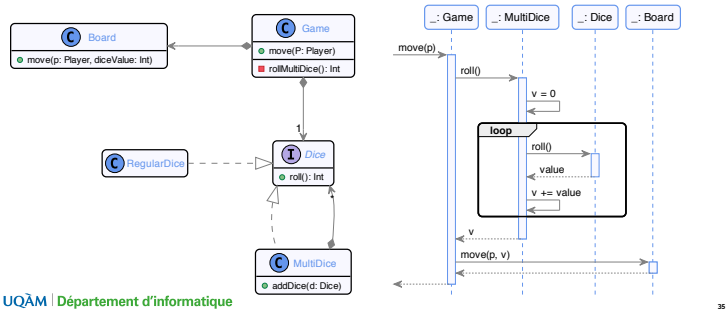
ENCART CAMÉRA






# Application au Monopoly

ENCART CAMÉRA






# Récapitulatif

ENCART CAMÉRA

- Permet de **maintenir un faible couplage et une forte cohésion** dans une application objet
- **Améliore la réutilisabilité** des éléments
- Repose sur une **entité créée** de toute pièces pour l'occasion

- **Questions à se poser** :
  - *Est-ce que quelqu'un pourrait porter cette responsabilité plus naturellement ?*
  - *Est-ce qu'on s'éloigne trop de la logique d'affaire ?*

# Avantages & Inconvénients

ENCART CAMÉRA

- **Bénéfices** :
  - Maintien faible couplage et forte cohésion
- **Limitations** :
  - **Il ne faut pas en abuser**, sinon le modèle objet n'est plus cohérent avec la logique d'affaire du système
  - **Très artificiel**, nécessite de la documentation sur le long terme



# Patron #8 : Indirection

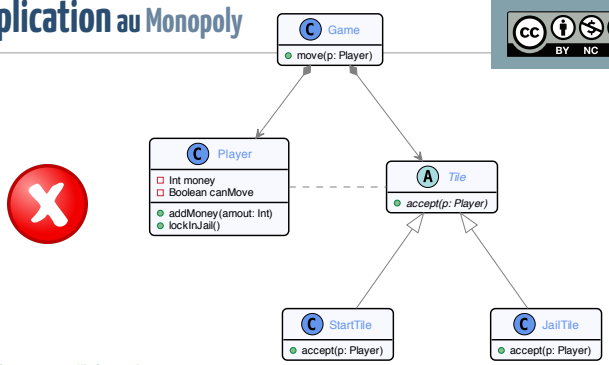
ENCART CAMÉRA

- **Situation** :
  - **Comment éviter un couplage immédiat entre plusieurs éléments ?**
- **Proposition** :
  - Introduire un élément dédié à ce couplage pour laisser les éléments pré-existants isolés
- **Exemple** :
  - **Comment faire en sorte de garder les Cases et les Joueurs indépendants ?**



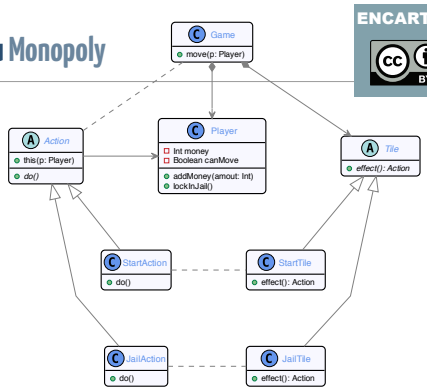
# Application au Monopoly

ENCART CAMÉRA



# Application au Monopoly








# Récapitulatif



- Permet de découpler des éléments du système
- Le découpage est parfois artificiel

- **Question à se poser** :
  - *Comment maintenir séparées ces deux entités ?*
  - *Est-il nécessaire de maintenir séparées ces deux entités*



# Avantages & Inconvénients



- **Bénéfices** :
  - Favorise un **couplage faible**
  - Permet la **co-évolution**
- **Limitation** :
  - **Complexifie la structure** du modèle objet
  - Rajoute un **coût à l'exécution** (passer par l'indirection)
  - "*Un architecte qui rate un bâtiment dira à son client de planter une vigne sur le mur pour le cacher.*
    - ***Un architecte logiciel ajoutera un niveau d'indirection***." - Booch 2019.

# Patron #9 : Protégé des Variations

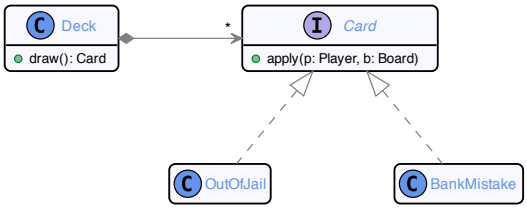


- **Situation** :
  - **Comment concevoir des objets qui ne seront pas impactés par les variations ou l'instabilité d'autres parties du système ?**
- **Proposition** :
  - Trouver ce qui varie et l'encapsuler dans une interface stable
- **Exemple** :
  - **Comment gérer les cartes spéciales au Monopoly ?**



# Application au Monopoly








# Récapitulatif



- Permet d'**anticiper les évolutions**
- Différents **niveaux de maîtrise de la Force** :
  - Le ***padawan*** conçoit du **code fragile**
  - Le ***chevalier*** conçoit de façon **souple** et **généralisante**
  - Le ***maître Jedi*** sait **choisir les batailles** à livrer
- **Question à se poser** :
  - *Est-ce que j'ai VRAIMENT besoin d'une protection ici ?*

# Avantages & Inconvénients

ENCART CAMÉRA

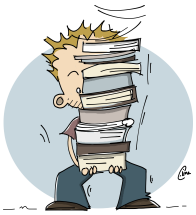
- **Bénéfices** :
  - Permet de **livrer du code évolutif** par construction
  - Les **points de variations sont identifiés** pour les futurs développeurs
- **Limitations** :
  - Attention à l'***over-engineering***
  - Protéger son code **prend du temps**



UQÀM | Département d'informatique

FACULTÉ DES SCIENCES
Université du Québec à Montréal

ENCART CAMÉRA



https://mosser.github.io/

ace

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

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