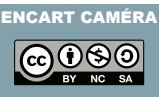





ENCART CAMÉRA


Pourquoi encapsuler ?
UQÀM | Département d'informatique

Sébastien Mosser - INF5153
 Chapitre 2 - Capsule 2
 Automne 2020


Pourquoi encapsuler ?


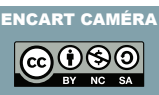
- On **décompose** un système en objets pour :
 - maîtriser la complexité** globale (*diviser pour mieux régner*)
 - travailler de manière séparée** sur les concepts (*conflits git*)
- Tout système décomposé est-il bon ?

Code Possible
 
 Objectif

 Conception Acceptable

Éviter au maximum le problème dit de "ignorant surgery"

UQÀM | Département d'informatique
2

Définition de l'encapsulation


- Regrouper** dans une "capsule" (p.-ex., une classe)
 - Des données brutes
 - Les traitements permettant de les manipuler selon la logique d'affaire
- Définir des **règles de visibilité**
 - Pour empêcher la manipulation des données hors logique d'affaire
 - Pour permettre d'assurer de l'intégrité

UQÀM | Département d'informatique
3

Un exemple de conception de cartes

ENCART CAMÉRA



- Description extraite depuis la **spécification** :

- Il y a neufs (9) cartes *clan* par couleurs
- Il y a six (6) couleurs de cartes

- On **encode** l'information dans un **entier** $card \in [0, 9 \times 6[$

- Pour trouver la **valeur** de la carte : On **divise** par neuf, plus un

```
public static int value(int card) { return (card / 9) + 1; }
```

- Pour trouver la **couleur** : on prend le **modulo** neuf

```
public static int color(int card) { return card % 9; }
```



Quels sont les problèmes ?

ENCART CAMÉRA



- Représentation** dans le **code** **différente** du **domaine**

- Vous jouez avec des cartes, ou avec des nombres entiers ?

- Représentation **couplée** à l'implémentation

- Dans tout le programme, on utilise le type `int` pour une carte

- On peut **corrompre** la structure de donnée aisément

- Est-ce que -42 est une carte valide ? Pourtant c'est un `int` ...

On parle de **"Primitive Obsession"** quand un code utilise de manière inadéquate des types primitifs au lieu d'objets du domaine. C'est une **représentation possible** dans le code, mais **pas acceptable** du point de vue de la conception.

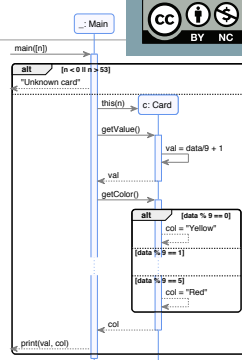
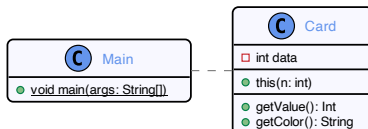


On va encapsuler alors !

ENCART CAMÉRA



Classes : ☒ Modèles UML : ☒
C'est facile la conception !



Quels sont les problèmes ?

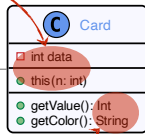
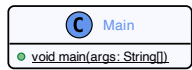


ENCART CAMÉRA



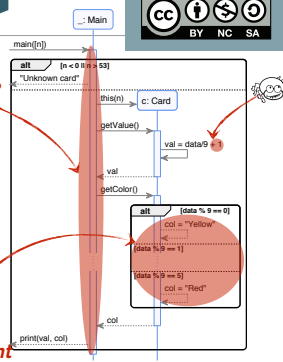
Choix interne visible

Responsabilité ?



Primitive Obsession

Switch statement



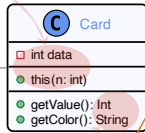
Un dernier problème pour la route

ENCART CAMÉRA



Choix interne visible

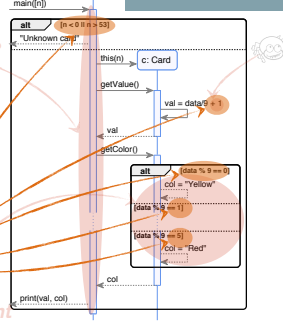
Responsabilité ?



Primitive Obsession

Nombres Magique

Switch statement



Cause racine & amélioration de la conception

ENCART CAMÉRA



- Pour améliorer notre conception, **il ne faut pas s'arrêter aux symptômes**
- La cause racine de nos problèmes ici est la **Primitive Obsession**
 - **En faisant le choix de reposer sur des *Ints* et des *Strings*, on fragilise tout le système**
- Pour palier ce problème, il faut "**réifier**" les concepts associés :
 - Une valeur de carte n'est plus un **Int** mais une **Value**
 - Une couleur de carte n'est plus une **String** mais une **Color**

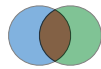
Réifier ... avec une classe, ou une énumération ?

ENCART CAMÉRA

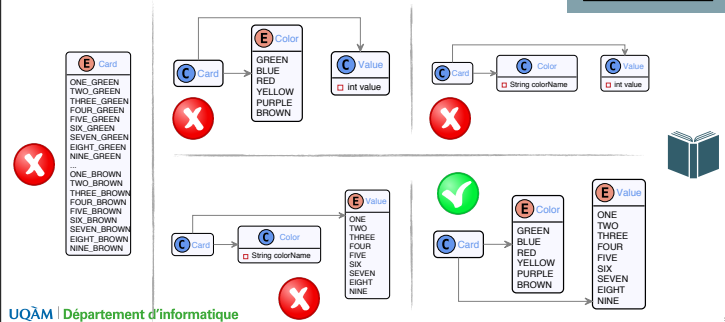


- La plupart des langages propose la **notion d'énumération**
 - Souvent avec le mot clé "enum"
- Quelle **différence** avec une classe ?
 - Une **énumération** repose sur l'**hypothèse du monde fermé**
 - = Extension (Ensemble des nombres pairs $< 10 = \{0, 2, 4, 6, 8\}$)
 - Une **classe**, à l'inverse, fonctionne en **monde ouvert**
 - = Intention (Ensemble des nombres pairs $= \{n \mid n \% 2 == 0\}$)
- Dans notre cas, **comment choisir entre classe & énumération** ?

Exploration de l'espace des conceptions



ENCART CAMÉRA

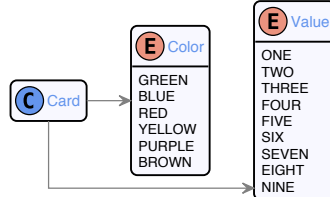


Code Java associé

ENCART CAMÉRA



```
class Card {  
    Value value;  
    Color color;  
}  
  
enum Value {  
    ONE, TWO, THREE, FOUR, FIVE,  
    SIX, SEVEN, EIGHT, NINE  
}  
  
enum Color {  
    GREEN, BLUE, RED, YELLOW, PURPLE, BROWN  
}
```



Renforcement de l'encapsulation

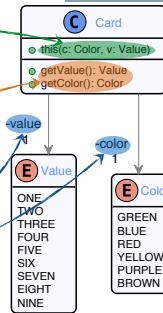
- On manipule maintenant dans le code des **concepts en lien direct avec la logique d'affaire**
- Les données sont protégées d'accès incontrôlés.
- Les get/set permettent de former des **propriétés**

Constructeur "domaine"

Données interne non modifiables (immuable)

Données interne privées

ENCART CAMÉRA

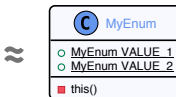
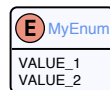


Pour aller plus loin : Énumérer avec des Classes

```
enum MyEnum {
    VALUE_1, VALUE_2
}
```

"presque équivalent"

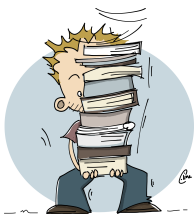
```
class MyEnum {
    public static final MyEnum VALUE_1 = new MyEnum();
    public static final MyEnum VALUE_2 = new MyEnum();
    private MyEnum() {}
}
```



"final" is ALL CAPS by convention in UML

On parle de "sucre syntaxique", une construction de langage qui rend facile d'accès quelque chose qui est déjà possible dans le langage.

ENCART CAMÉRA



<https://mossergithub.io/>



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

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

