ENCART CAMÉRA

Types & Interfaces

Sébastien Mosser - INF5153
Chapitre 2 - Capsule 4
Automne 2020

ENCART CAMÉRA

Quelle est la différence entre un Type et une Classe ?

- Un "**Type**", c'est :
  - *"une annotation qui aide a garder le code cohérent" (Privat 2019)*
  - un contrat entre consommateur et fournisseur de service
- Une "**Classe**" :
  - "c'est *une capsule qui décrit des objets similaires*" (*Privat 2019*)
  - *peut être conforme à plusieurs **types***.

```
public void List<Card> getCards() {
  List<Card> result = new ArrayList<>();
  for(Card c: this.cards) {
    Card copy = new Card(c.getColor(), c.getValue());
    result.add(copy);
  }
  return result;
}
```

ENCART CAMÉRA

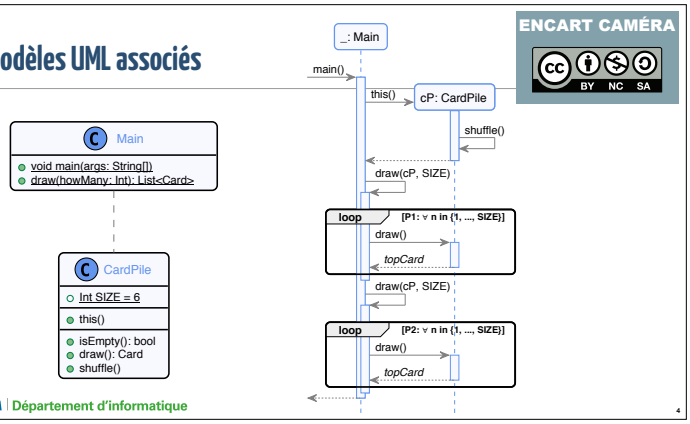
Distribuer les cartes aux joueurs

- Pour piger 'n' cartes dans une pile de cartes, il faut :
  - ***Savoir s'il en reste une*** dans la pile
    - *par exemple avec un message "isEmpty?: void → Bool"*
  - ***En piger une*** dans la pile
    - *par exemple avec un message "draw: void → Card"*

Du code bien conçu doit refléter la logique d'affaire

```
public static List<Card> drawCards(CardPile cp, int howMany) {
  List<Card> result = new ArrayList<>();
  for (int i = 0; i < howMany; i++) {
    if (cP.isEmpty()) {
      return result;
    }
    result.add(cP.draw());
  }
  return result;
}
```

Modèles UML associés



Responsabilités & Réutilisabilité

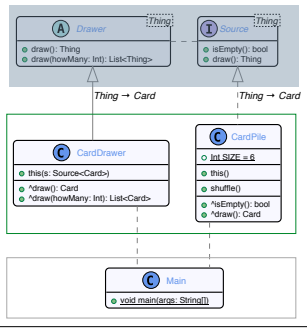
- On a donné à "Main" la responsabilité de piger les cartes
  - **C'est pas fou d'un point de vue logique d'affaire**
- Est-ce que **CardPile** pourrait détenir cette responsabilité ?
  - Oui, il semblerait plus logique que ce soit la pile qui donne son contenu, d'un point de vue encapsulation
- Et si on veut **distribuer autre chose que des cartes Schotten Totten** ?
  - Auquel cas il faut réifier un concept lié à la notion de *distribution*

Allons avec un distributeur de "trucs"

On pige un "truc" dans une "source"

On pige une carte dans la pile

Le jeu Schotten Totten



ENCART CAMÉRA

Couplage Faible

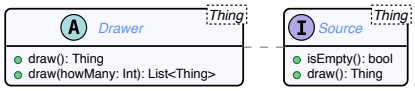
Réutilisabilité

ENCART CAMÉRA

Choisir entre Classe Abstraite & Interface

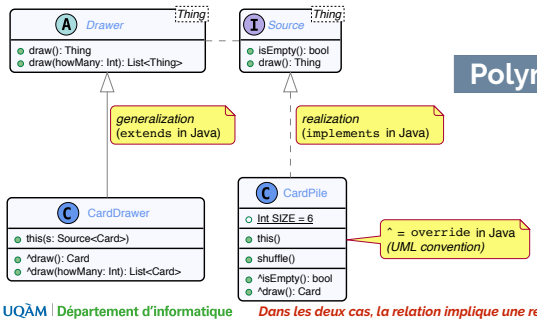
- Une **Interface** :
  - *Ensemble de services fournis par les entités conforme à ce Type*
- Une **Classe Abstraite** :
  - *Comportement abstrait que l'ont sait décrire en l'état*
- *Et une interface avec des **méthodes "par défaut"** ?*
  - *C'est avant tout une interface (et du sucre syntaxique)*

__Drawer__ est une classe __Abstraite__ :
On sait décrire son comportement en l'état des services offerts par __Source__



ENCART CAMÉRA

Relations de Généralisation et de Réalisation



Polymorphisme

 Dans les deux cas, la relation implique une relation de sous-typage 9

Généralisation versus Réalisation

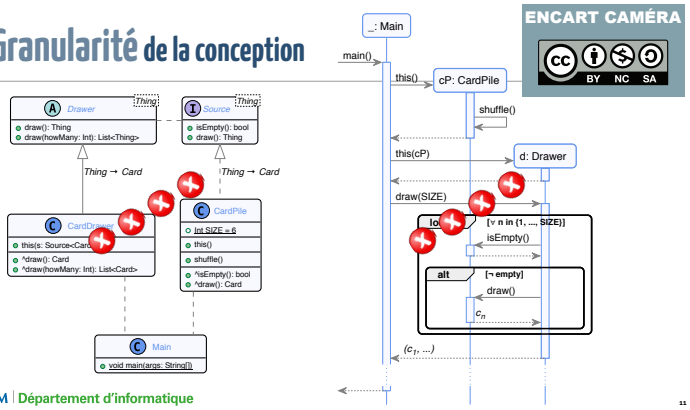
ENCART CAMÉRA

- La **généralisation** est une relation **taxinomique**
  - Taxinomie = **classification** *(p.-ex. les espèces en biologie)*
  - Relation entre un élément ***général*** et un autre ***spécifique***
    - *Un chat est un mammifère. Une carotte est un légume.*
- La **réalisation** est une relation de **mise en oeuvre**
  - C'est un lien entre une **spécification** et sa **réalisation**
  - *Une collection sait se trier. Une pioche sait piger une carte.*

En pratique vous utilisez usuellement très mal ces deux relations. On parlera d'héritage (généralisation) dans le chapitre quatre.

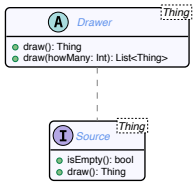
Granularité de la conception

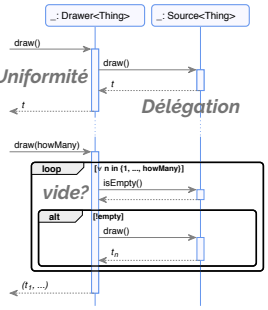
ENCART CAMÉRA



Faire plusieurs modèles différents (points de vues)

ENCART CAMÉRA

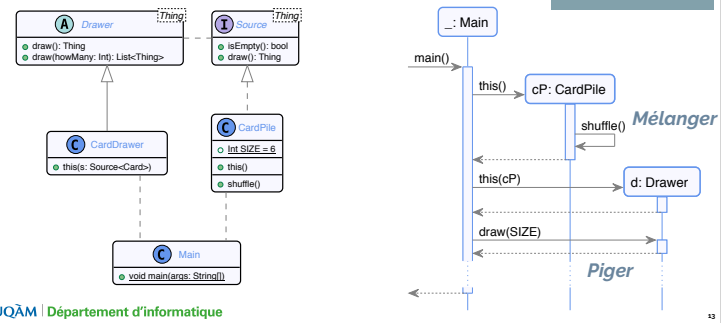




*Fait apparaitre un problème de conception : Oui, l'interface est uniforme pour piger, mais la **gestion de la source vide est différente** !*

Autre point de vue : Distribution des cartes

ENCART CAMÉRA



Questions de conception à se poser

ENCART CAMÉRA

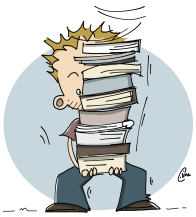
- *Est-ce que j'ai besoin de créer une nouvelle interface ?*
  - *Ça dépend.* 😇
- *Que devrait spécifier cette interface ?*
  - *Ça dépend.* 😇

- Ne pas tomber dans le piège de l'***over-engineering***
  - Ne pas créer trop de ***dette technique*** non plus
- C'est un processus **essais-erreurs** ⊕ **expérience**
  - *Les solutions dogmatique sont souvent inutilisable en pratique*

UQÀM | Département d'informatique

FACULTÉ DES SCIENCES
Université du Québec à Montréal

ENCART CAMÉRA



https://mosser.github.io/

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

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