

État des objets & Absence de valeur

Sébastien Mosser - INF5153
Chapitre 3 - Capsule 3
Automne 2020

UQÀM | Département d'informatique

créativité, photos, Pixabay

ace

L'état interne des objets modifie leurs réponses

- Via l'encapsulation, on a **caché à l'intérieur des objets** ce qui ne doit pas être visible
- **Ces données modifient naturellement les réponses** retournées par les objets quand on leur envoie des messages
- **Les objets passent donc d'un état à un autre**
 - Par exemple : **Disponible**, **Indisponible**, **Vide**, **Plein**, ...
- **Concevoir ces états est crucial** pour la réutilisabilité des objets

UQÀM | Département d'informatique

Comment s'intéresser à l'état des objets ?

- Les concepts manipulés ont de **grands ordres de grandeur**
- Considérons la classe **Player**, qui contient le score du joueur
 - Le score est un Int, **on peut créer 2^{32} instances ≠ de Player !**
- Et si on rajoute le nom du joueur ?
 - Le **nombre d'instances ≠ devient infini** (*limité par la mémoire*)

C

Player
int score

→

C

Player
int score
String name

UQÀM | Département d'informatique

L'exemple de la pile de carte

ENCART CAMÉRA



- On considère la pile de cartes de **Schotten Totten**
- Il y a $A_k^n = \frac{n!}{(n-k)!}$ **arrangements sans répétitions** de k cartes parmi n dans la pile de cartes (ordonnée)
- La pile peut contenir de zéro (0) à cinquante quatre (54) cartes
- Donc on a modélisé $N = \sum_{k=0}^{54} A_k^{54}$ **états possible** de la pioche
 - $N = 6,27 \times 10^{71}$ **pioches différentes !**
 - On estime à 10^{80} **le nombre d'atomes dans l'univers**

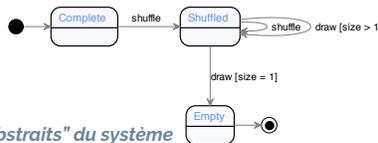


On considère uniquement les états pertinents

ENCART CAMÉRA



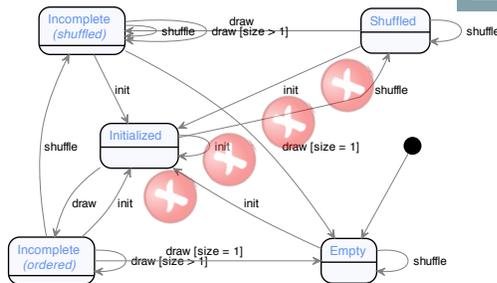
- L'utilité peut être à deux niveaux :
 - Technique** : "piocher une carte dans la "pile vide" ?"
 - Logique d'affaire** : "la pile de cartes est mélangée"



- C'est une modélisation des états "abstrait" du système
 - Ça n'a de sens que pour les objets **avec états** et **mutables**

Attention à la granularité des modèles

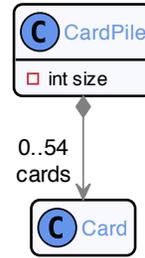
ENCART CAMÉRA



Problèmes classiques de conception

- Utiliser un State Diagram comme un **flot de données**
 - Un état n'est pas une étape de calcul sur des données !*
- Prévoir beaucoup d'états inutiles
 - Over-engineering**, ou encore "**Speculative Generality**"
 - Ça peut toujours servir ... Non ! (Dette technique)*
- Augmenter l'espace d'état pour "**optimiser**"
 - Souvent l'optimisation est minime*
 - C'est source de bugs dans le futur*
- Problème appelé "**Temporary Fields**"

ENCART CAMÉRA



Un état particulier : la nullité

- Comment **représenter l'absence d'information** ?
 - C'est le même problème que celui du "zéro" en math !*
 - Il faut une information concrète** représentant l'absence d'information ...
- En Java, pour représenter l'absence d'un élément de type T :**
 - Utiliser `null`
 - Concevoir** notre propre **zéro** pour le type en question:
 - p.-ex. une valeur spéciale, un objet dédié.*
 - Utiliser un `Optional<T>`

ENCART CAMÉRA



Exemple : La carte spéciale du Joker

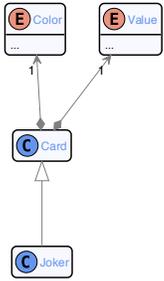
- La carte spéciale **joker** remplace **n'importe quelle carte clan** lors d'une attaque.
- On **choisit son contenu** au moment où on la pose.
 - Elle n'a donc ni Value ni Color jusqu'à ce qu'elle soit jouée.*
- Comment faire avec la conception actuelle ?



Le joker est une carte spéciale : HÉRITAGE !



ENCART CAMÉRA



**Exception "runtime"
viole la substituabilité (prochain cours)**

```
public class Joker extends Card {
    public Color getColor() {
        throw new UnsupportedOperationException();
    }
    public Value getValue() {
        throw new UnsupportedOperationException();
    }
}
```

Utiliser "null" pour représenter le "zéro"



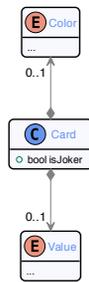
ENCART CAMÉRA



```
public class Card {
    private Value value = null;
    private Color color = null;
    private boolean isJoker = false;
    public Card(Value v, Color c) {
        this.value = v;
        this.color = c;
    }
    public Card() {
        this.isJoker = true;
    }
}
```

NullPointerException

- On rajoute un attribut booléen "isJoker"
- Le constructeur vide met null dans les champs value et color



Choisir une valeur nulle



ENCART CAMÉRA



```
public class Card {
    private Value value;
    private Color color;
    private boolean isJoker;
    public Card(Value v, Color c) {
        this.value = v;
        this.color = c;
        this.isJoker = false;
    }
    public Card() {
        this.value = Value.ONE;
        this.color = Color.BROWN;
        this.isJoker = true;
    }
}
```

Usurpation d'identité

*Et si on oublie de tester
si c'est un joker ?
Ça sera un "1 marron".*

- Le joker est un "1 marron". Parce que c'est comme ça.



Créer la valeur nulle



ENCART CAMÉRA



```
public class Card {
    private Value value;
    private Color color;
    private boolean isJoker;

    public Card(Value v, Color c) {
        this.value = v;
        this.color = c;
        this.isJoker = false;
    }

    public Card() {
        this.value = Value.NA;
        this.color = Color.NA;
        this.isJoker = true;
    }
}
```

Technique versus Affaire

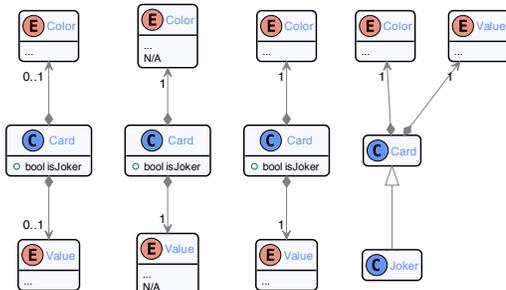
Pour l'initialisation de la pile de cartes on bouclait sur les énumérations ...



- On rajoute une valeur "N/A" dans l'énumération

Différences subtiles, mais gros impacts !

ENCART CAMÉRA



Gérer l'absence de valeur avec un Optionnel

ENCART CAMÉRA



- Construction présente dans beaucoup de langages
 - P.-ex. Python, Haskell, Scala, ...
- Pourquoi **null** en Java de 1995 à 2014 ? (19 ans !!)
 - "I call it my **billion-dollar** mistake. It was the invention of the null reference in 1965... I couldn't resist the temptation to put in a null reference, simply because it was so easy to implement."
 - *Tony Hoare*, London QCon 2009
- Définition du **type optionnel** : $Optional<T> = T \cup None$
 - Le **type Optionnel<Int>** représente n'importe quel **Int**, ou **None**

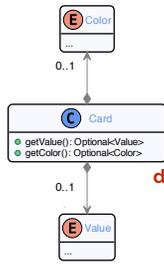
Optionnels visible

```
public class Card {
    private Value value;
    private Color color;

    public Card() {
        this.value = null;
        this.color = null;
    }

    public boolean isJoker() {
        return this.getValue().isPresent();
    }

    public Optional<Value> getValue() {
        if (this.value == null)
            return Optional.empty();
        else
            return Optional.valueOf(this.value);
    }
}
```



Fuite
d'abstraction ?



<https://mosser.github.io/>



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

Abonne toi à la chaîne,
et met un pouce bleu !

