



Principes SOLID & Patrons GRASP

UQÀM | Département d'informatique

Crédit Images: Pixabay & Pexels

Sébastien Mosser
INF 5153 - Cours #4 - A19



1

Bibliographie du cours



2

Crédits

UNIVERSITÉ CÔTE D'AZUR



Mireille Blay-Fornarino

Université Côte d'Azur
Laboratoire I3S, SPARKS

Imbattable au sprint, même avec des talons

<http://mireilleblayfornarino.i3s.unice.fr/>

3

Crédits

UNIVERSITÉ CÔTE D'AZUR



Philippe Collet

Université Côte d'Azur
Laboratoire I3S, SPARKS

"Prof d'UML" de père en fils depuis 1999

https://www.i3s.unice.fr/Philippe_Collet

4



5



7

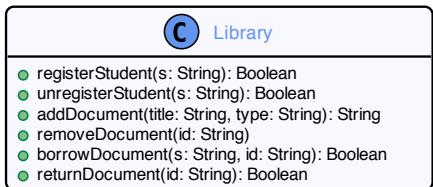


6



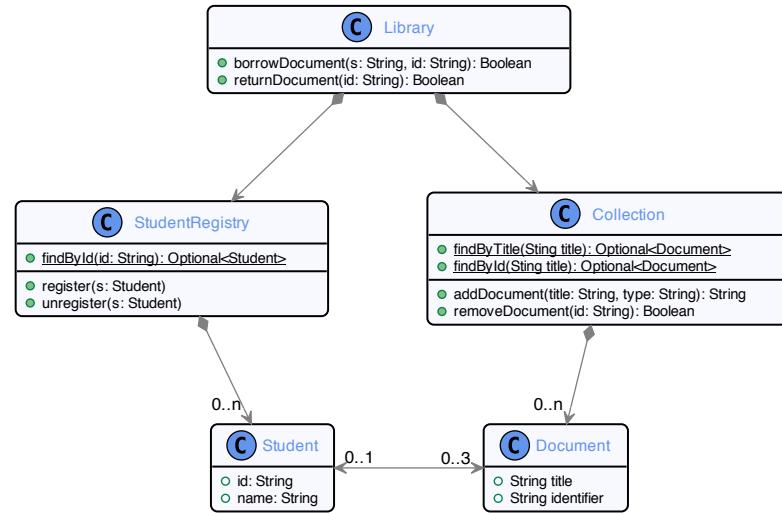
8

Responsabilité Unique (SOLID)



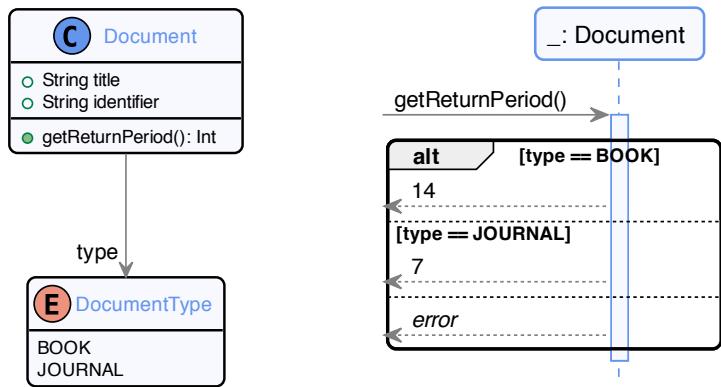
9

Responsabilité Unique (SOLID)



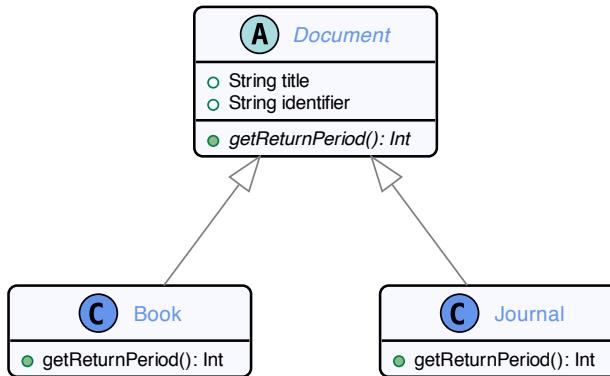
10

Principe Ouvert/Fermé (SO_{LID})



11

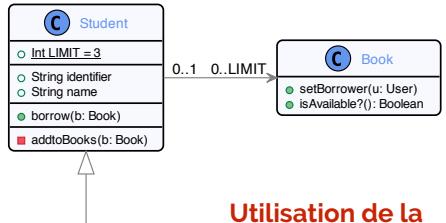
Principe Ouvert/Fermé (SO_{LID})



Polymorphisme >> *switch statement*

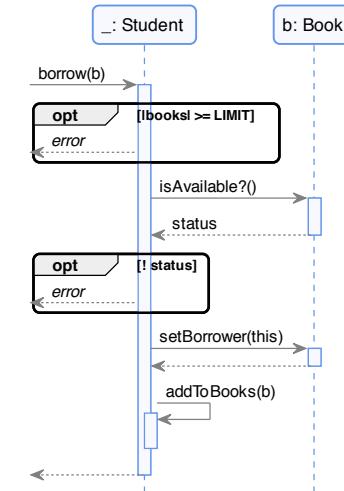
12

Substitution de Liskov (soLID)



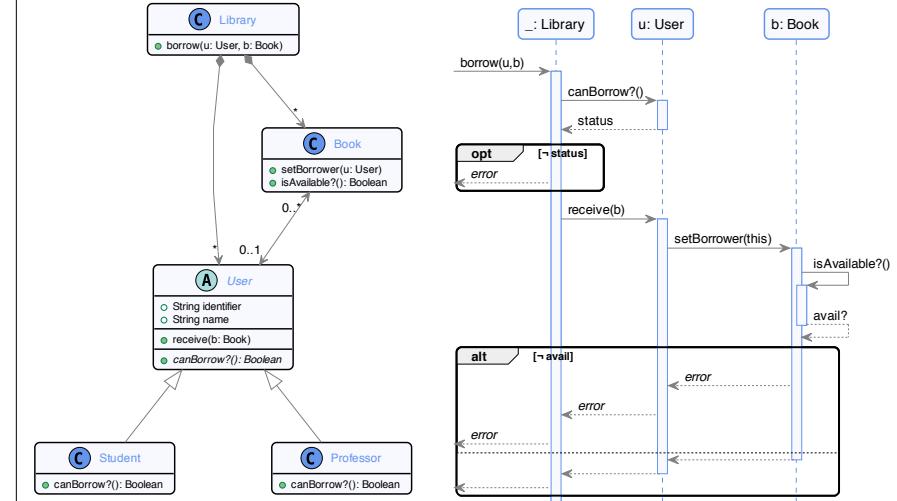
Utilisation de la généralisation pour de la réutilisation seulement, sans taxinomie.

"Un professeur EST UN étudiant"
Vraiment ???



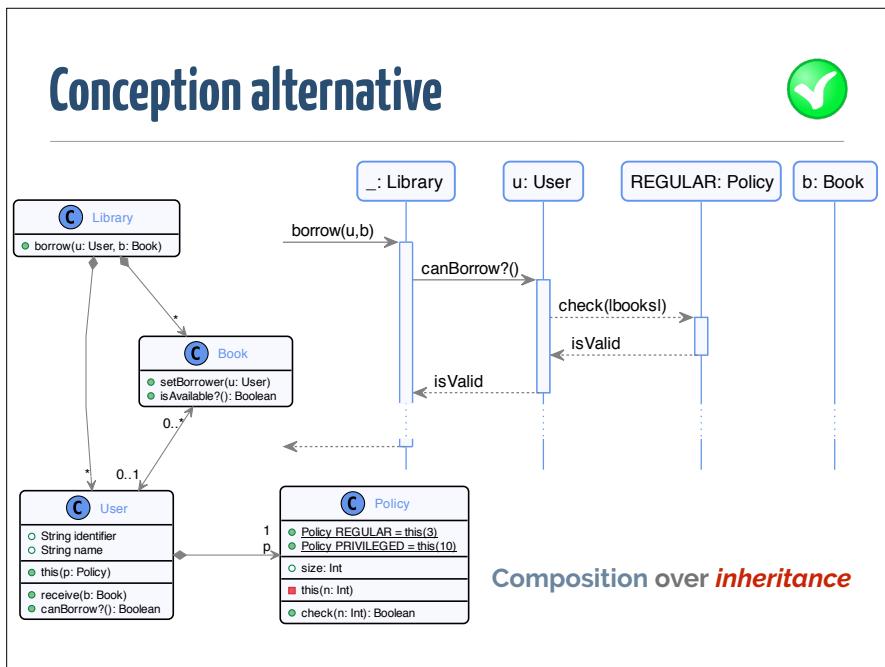
13

Substitution de Liskov (soLID)



14

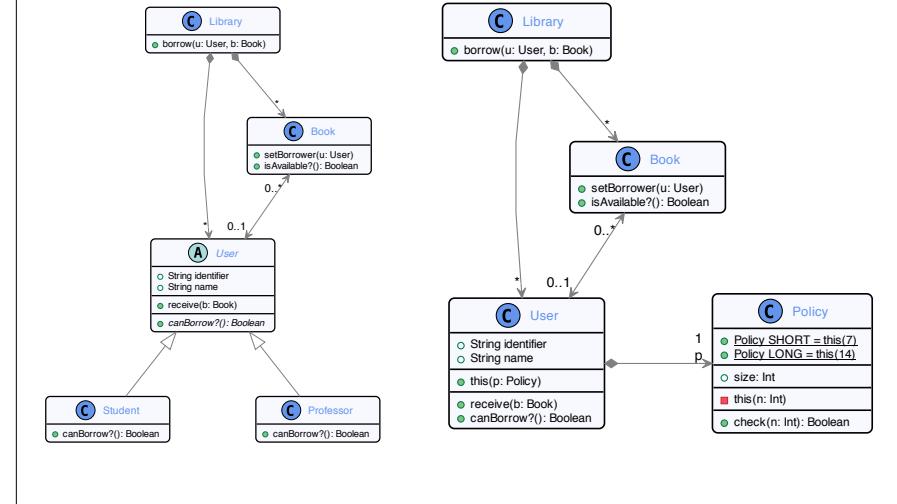
Conception alternative



Composition over inheritance

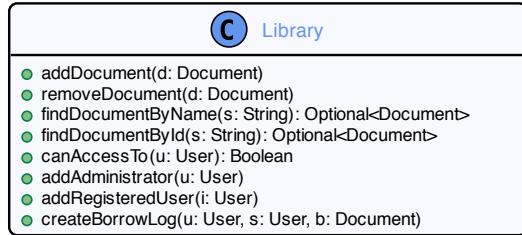
15

Question d'Examen : analyser / comparer



16

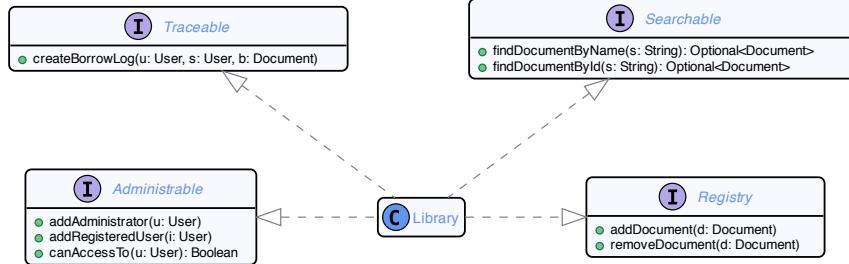
Ségrégation des Interfaces (sOlid)



On exploze aussi le principe S ici ...

17

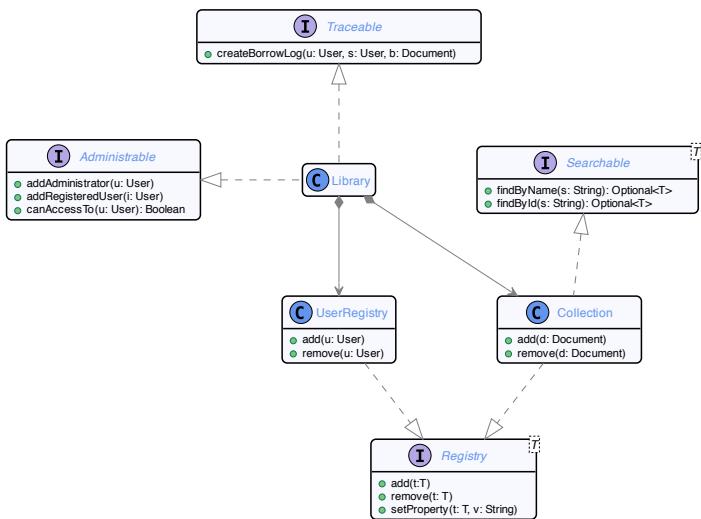
Ségrégation des Interfaces (sOlid)



On exploze TOUJOURS le principe S ...

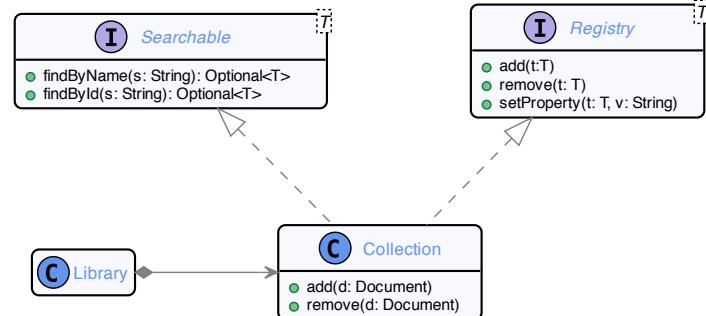
18

Ségrégation des Interfaces (sOlid)



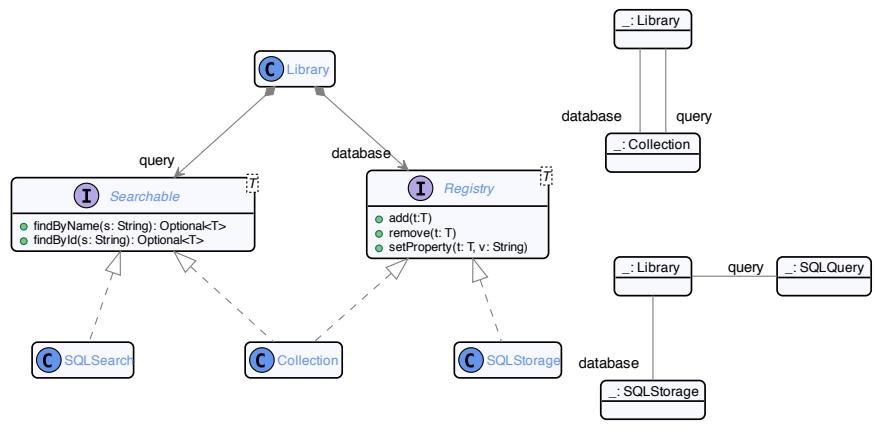
19

Injection des Dépendances (sOlid)



20

Injection des Dépendances (soliD)



21

Patrons GRASP



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

22

Problèmes avec SOLID

- **Les principes SOLID ne sont QUE des "principes"**
 - Pas de méthodologie pour les appliquer systématiquement
 - Une méthodologie ne résoudrait pas le problème en fait ...
 - Parce que comme toujours, "**ça dépend**"
 - On va plutôt utiliser une approche par "**Patrons**" (patterns)
 - Un **patron** est une **solution classique** à un problème récurrent, ayant plus **d'avantages** que **d'inconvénients** pour résoudre ce problème.
 - Il faut bien **identifier le problème**, et souvent **adapter le patron**



23

24

L'approche GRASP



- GRASP est une collection de **9 patrons de responsabilisation**
 • *General Responsibility Assignment Software Patterns*
- Ce sont des **aides à la conception** d'application objet
 - Pour guider la conception
 - Pour rationaliser la conception
 - Pour rendre explicable la conception
- Les patrons GRASP reposent sur le principe de **Responsabilité**
- L'objectif est de **réduire le décalage entre code et logique d'affaire** en pensant le logiciel **en terme de responsabilité**.

25

Typiquement ...

- Les questions suivantes **NE SONT PAS** des questions d'examens :
 - "Citez les neufs patrons GRASP"
 - "Citez les cinq principes SOLID"
 - "Définissez l'acronyme BICEP"
- Les questions suivantes **SONT** des questions typiques :
 - "Expliquez le rôle du patron GRASP 'Pure Invention' dans le cadre d'une conception orientée objet"
 - "Définissez en vos propres termes le principe de Substitution de Liskov (L de SOLID) et expliquez son utilité lors de la conception d'une hiérarchie de types"

Parce que le but de ce cours est de vous donner des armes pour votre vie de développeur professionnel.

27

Liste des patrons GRASP

- Spécialiste de l'Information
- Créateur
- Faible Couplage
- Contrôleur
- Forte Cohésion
- Polymorphisme
- Pure Invention
- Indirection
- Protégé des variations

Ce sont des patrons "naturels" qui cristallisent du bon sens.

Le but n'est pas de les apprendre par ❤ et de savoir réciter leur description, mais de comprendre l'intention sous-jacente et de l'appliquer dans vos conceptions.

26

Qu'est-ce qu'une responsabilité

- Une **responsabilité** est une abstraction de comportement
 - Ce n'est pas une méthode
 - Les méthodes s'acquittent des responsabilités
- **Différents types** de responsabilités :
 - La responsabilité de **FAIRE les choses**
 - P.-ex., déclencher une action
 - La responsabilité de **SAVOIR les choses**
 - P.-ex., Connaitre ce qu'un objet peut calculer

28

Un objet CONNAIT des choses

- Connaitre les **valeurs de ses propriétés**
 - Encapsulation des données*
- Connaitre les **objets qui lui sont rattachés**
 - Encapsulation, fuite de données, décision de copie, ...*
- Connaitre les **données qu'il peut dériver**
 - P.-ex, la taille d'une collection*

Il est de la responsabilité du développeur de gérer cette connaissance en respectant les principes d'encapsulation

29

Patron #1 : Spécialiste de l'Information



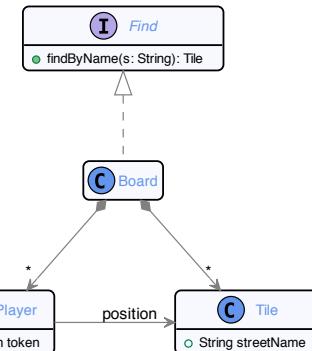
- Situation :**
 - A qui donner une responsabilité ?**
- Proposition :**
 - Donner la responsabilité à la classe qui **connaît** les informations permettant de **faire** cela.
- Exemple :**
 - Au Monopoly, on a des **joueurs** qui se déplacent sur des **cases** disposées sur un **plateau** de jeu.*
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

Un objet FAIT des choses

- Il **rend des services** (via des invocations de méthode)
 - P.-ex. En faisant un calcul, en fabriquant un nouvel objet*
- Il **délègue à un objet** sachant faire
 - "Je ne sais pas faire, mais lui il sait faire" (cf. composition)*
- Il **coordonne les actions** des autres objets
 - Je demande au meuble multimédia de se déverrouiller*
 - Si l'action à échouée, je contacte le service de l'audiovisuel pour un déverrouillage distant*
 - Je bascule le local en mode "présentation"*

30

Application au Monopoly



Le plateau **connaît** les cases,
il est le **SPÉCIALISTE** pour en **faire la recherche** par nom

31

32

Récapitulatif

- Patron le plus utilisé pour affecter les responsabilités
 - 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 ?

33

Patron #2 : Créeur



- 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 a des données permettant d'initialiser les instances de C'
- Exemple :
 - Qui crée les cases de jeu au Monopoly ?

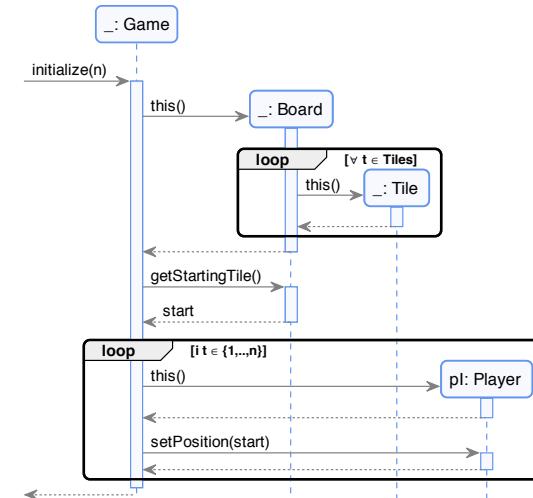
35

Avantages & Inconvénients

- 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 nécessaire soit partagée entre différents objets.

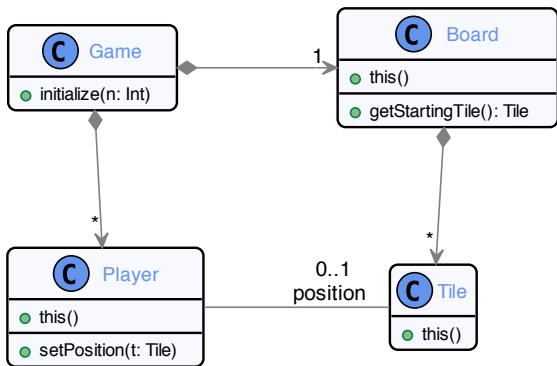
34

Application au Monopoly



36

Question d'Examen : Diagramme de classe ?



37

Avantages & Inconvénients

- 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 recycler des objets (*p.-ex. avec un bassin*)
- Limitation :
 - Pas toujours évident quand les objets sont partagés
 - Problème des liens bidirectionnels entre objets

39

Récapitulatif

- 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"

38

Patron #3 : Faible Couplage



- 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

40

Exemples de Couplage

- Un type **X est couplé à** un type **Y** quand :
 - X à un **attribut de type Y** (composition)
 - X à 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 séquences **concentrent les envois de messages** vers d'autres objets

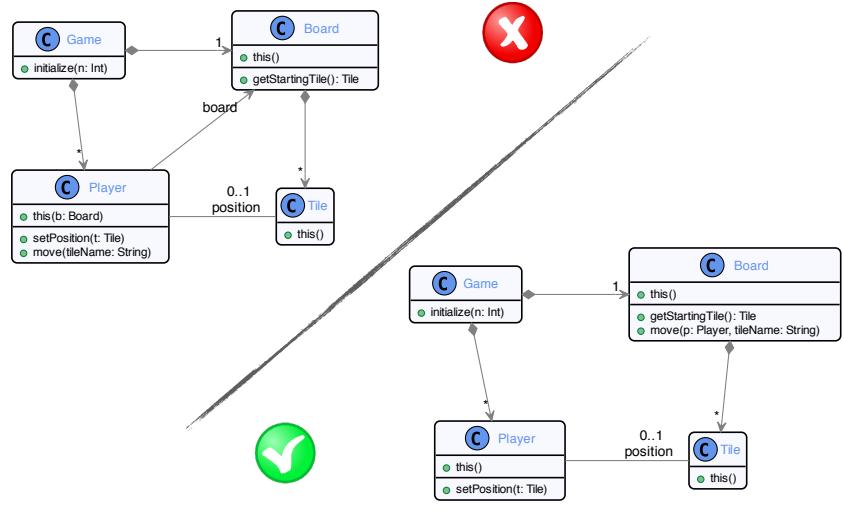
41

Récapitulatif

- Un **couplage fort** force à **changer tout ou 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 à VRAIMENT besoin de connaître celui-ci ?*

43

Application au Monopoly



42

Avantages & Inconvénients

- **Bénéfices :**
 - Les classes faiblement couplé sont **facile à comprendre**
 - **Facilité d'écriture** des tests
- **Limitations :**
 - Si tout le système est découpé, les **objets sont incohérents** et complexe, regroupement trop de responsabilités
 - **Il est facile d'escroquer la métrique de couplage** pour la faire passer pour faible (p.-ex. en stockant l'identifiant d'un objet comme un entier plutôt que de référencer l'objet en question)
 - Ça se voit ... et c'est pire

44

Patron #4 : Contrôleur



- 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 ?

45

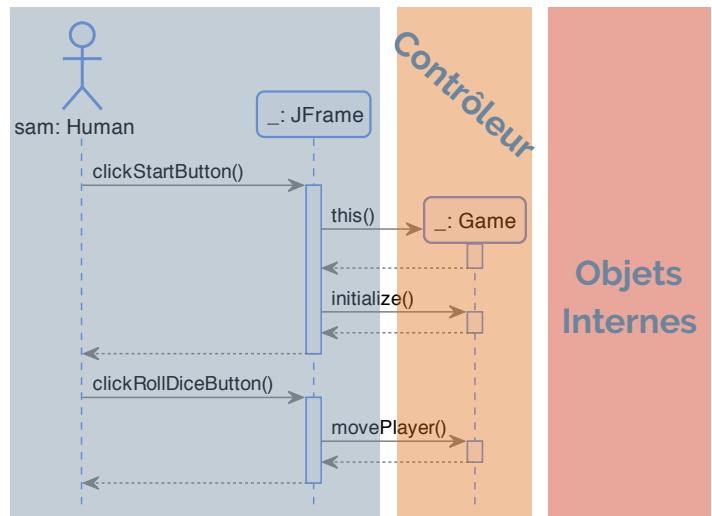
Récapitulatif

- Si vous avez entendu parlé des architectures suivant 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)

¹ Les architectures actuelles utilisent plutôt MVVM, une évolution de MVC

47

Application au Monopoly



46

Avantages & Inconvénients

- 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.

48



49

Patron #5 : Forte Cohésion



Situation :

- Comment s'assurer que les objets restent compréhensible et facile 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 connaître l'état du jeu ?

Patrons GRASP

le retour

50

Cohésion ?

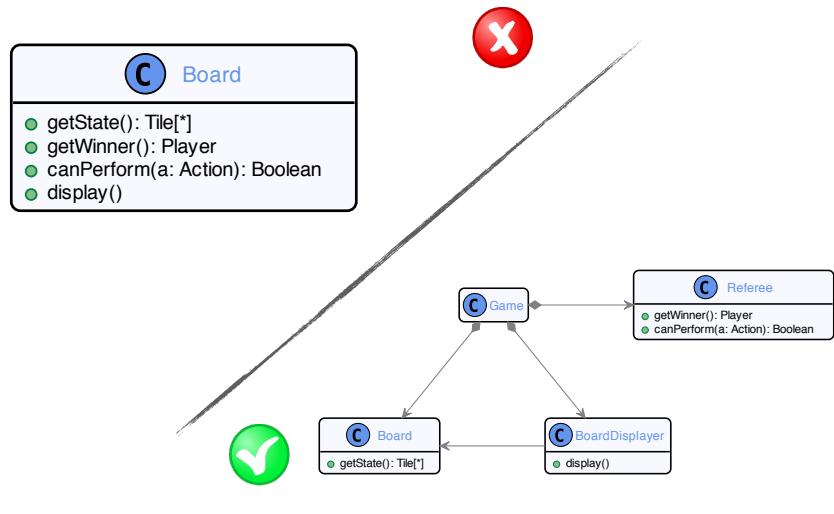
- La cohésion est une **mesure de l'étroitesse des liens** et de la spécialisation des responsabilités d'un élément
- Une classe qui a des **responsabilités étroitement liées** les unes aux autres et qui **n'effectue pas un travail gigantesque** est fortement cohésive
- "Un objet bien conçu renferme 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 imaginé"
- Donald Norman, 1994
- Pas de définition "formelle". **Ça dépend.**

On cherche à maximiser la cohésion et à minimiser le couplage

51

52

Application au Monopoly



53

Avantages & Inconvénients

- **Bénéfices :**
 - Maintenance et évolutivité améliorés
 - 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"

55

Récapitulatif

- 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 ?

54

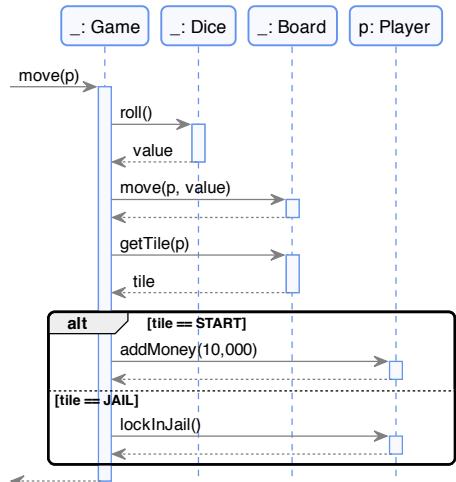
Patron #6 : Polymorphisme



- **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 inter-changées.
- **Exemple :**
 - Comment gérer les cases de jeu différentes au Monopoly ?

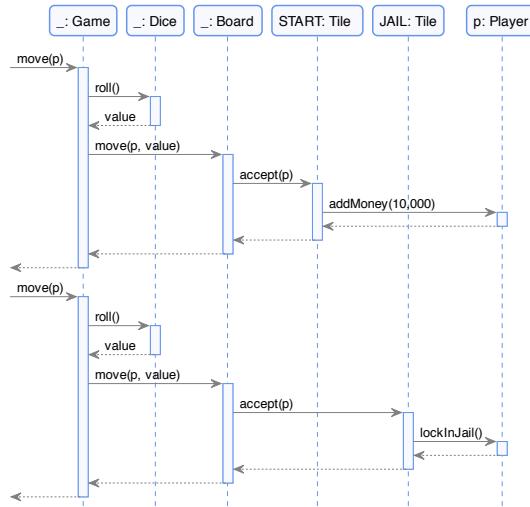
56

Application au Monopoly



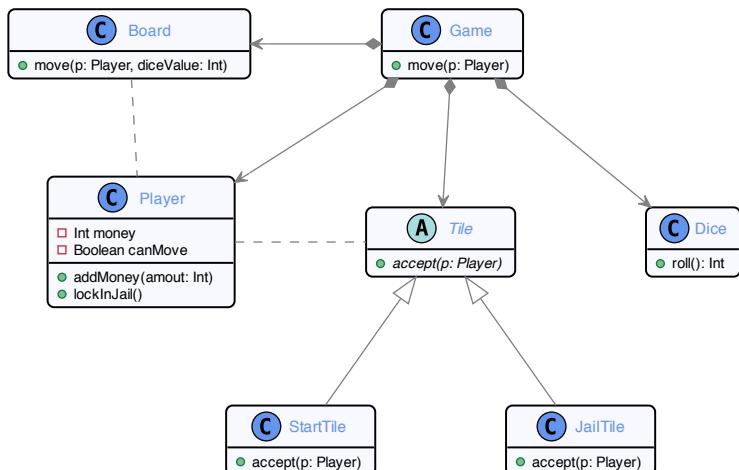
57

Application au Monopoly



58

Application au Monopoly



59

Récapitulatif

- 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 manière de réaliser ce service ?
 - Est-ce que ça dépend du type, ou de l'instance ?

60

Avantages & Inconvénients

- 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 existant
- Limitations :
 - Signatures polymorphes parfois **difficile à identifier**
 - Certains langages **limitent la hiérarchie** d'héritage

61

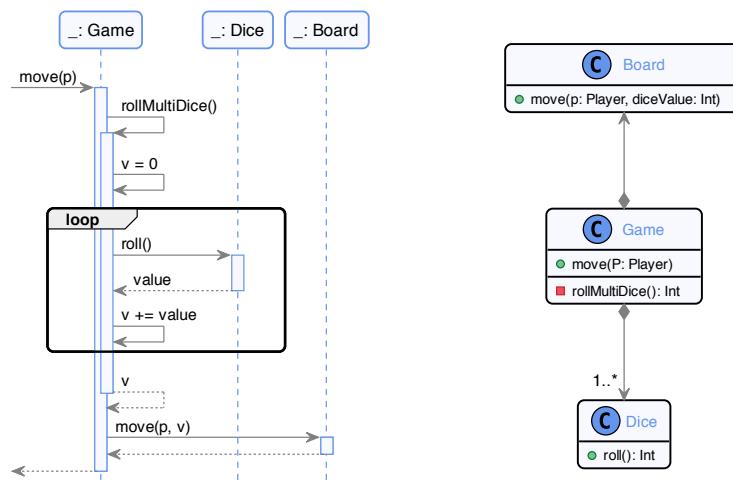
Patron #7: Fabrication Pure



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

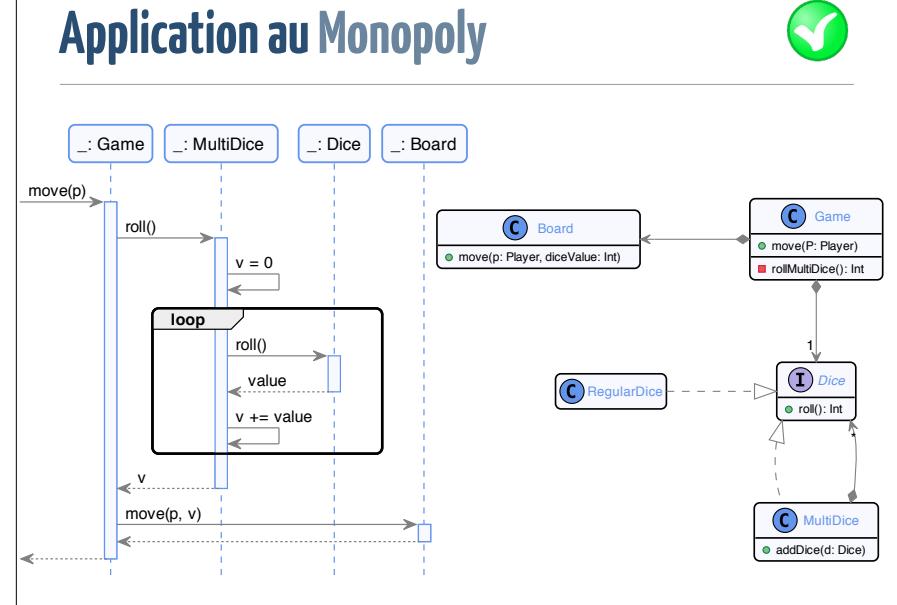
62

Application au Monopoly



63

Application au Monopoly



64

Récapitulatif

- 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
- Question 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 ?*

65

Patron #8: Indirection



- 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 ?**

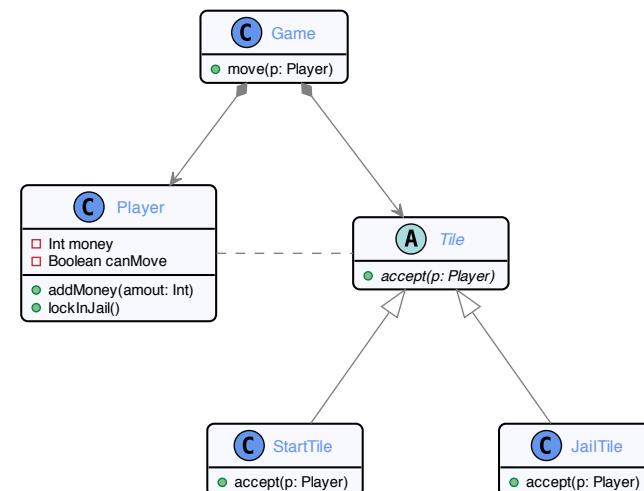
67

Avantages & Inconvénients

- 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

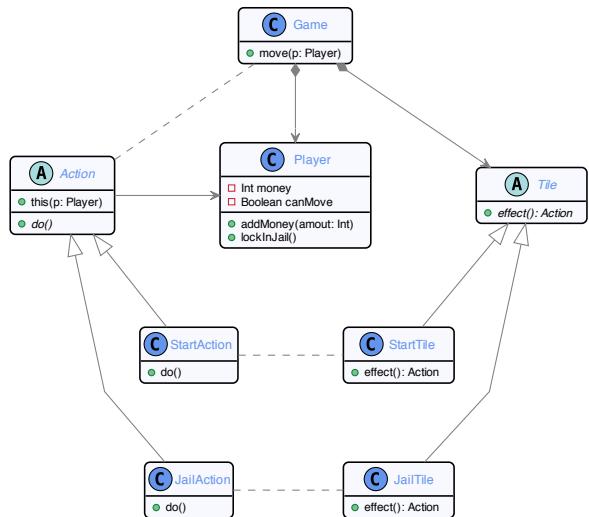
66

Application au Monopoly



68

Application au Monopoly



69

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.

71

Récapitulatif

- Permet de découpler des éléments du système
- Le découpage est parfois artificiel
- **Question à se poser :**
 - Comment maintenir séparés ces deux entités ?
 - Est-il nécessaire de maintenir séparée ces deux entités

70

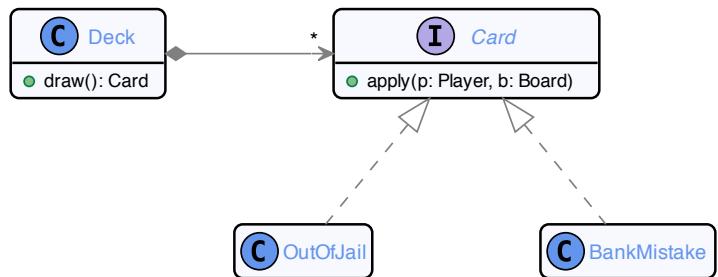
Patron #9: Protégé des Variations



- **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**

72

Application au Monopoly



73

Avantages & Inconvénients

- 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**

75

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 ?*

74

Étude de Cas (gestion de projet)



76

Spécifications

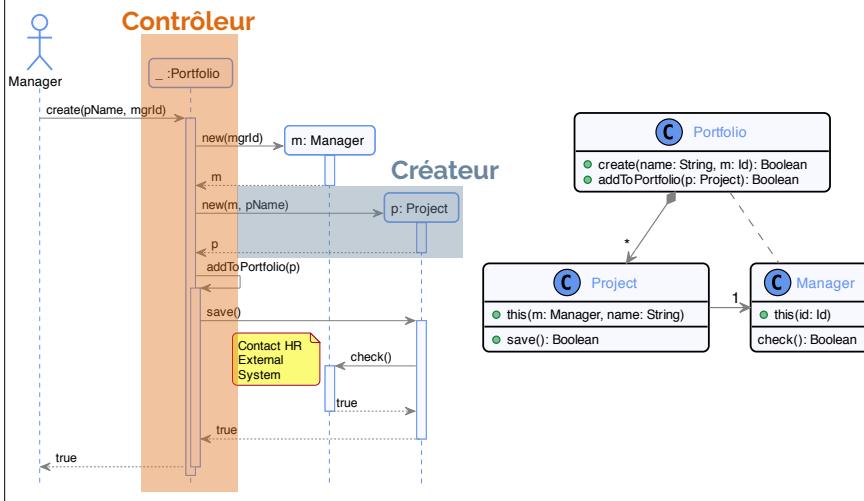
- On souhaite développer le **système de gestion des projets** interne à la société;
- Un **Manager** peut **créer des projets**, et **affecter des tâches** aux membres des projets;
- Les **membres** peuvent **entrer leurs progrès** sur une tâche du projet
- A tout moment, on peut :
 - Créer une nouvelle tâche** dans un projet ;
 - Générer un rapport d'avancement** au format PDF

77

La conception est une activité incrémentale

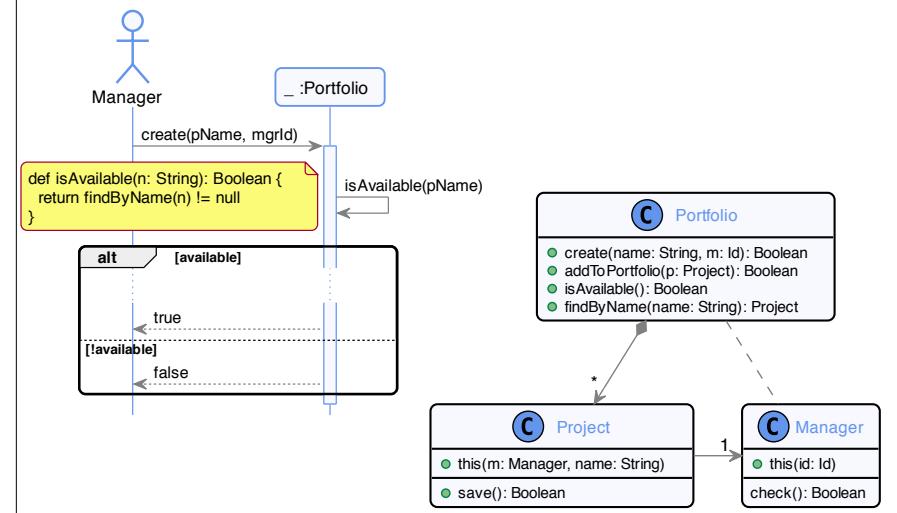
78

Fonctionnalité #1 : Créer un nouveau projet



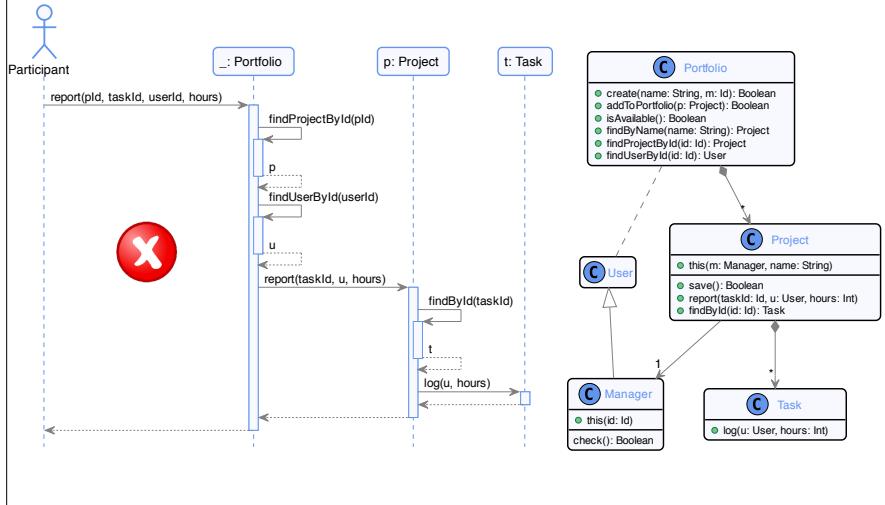
79

Fonctionnalité #2 : Disponibilité du nom ?



80

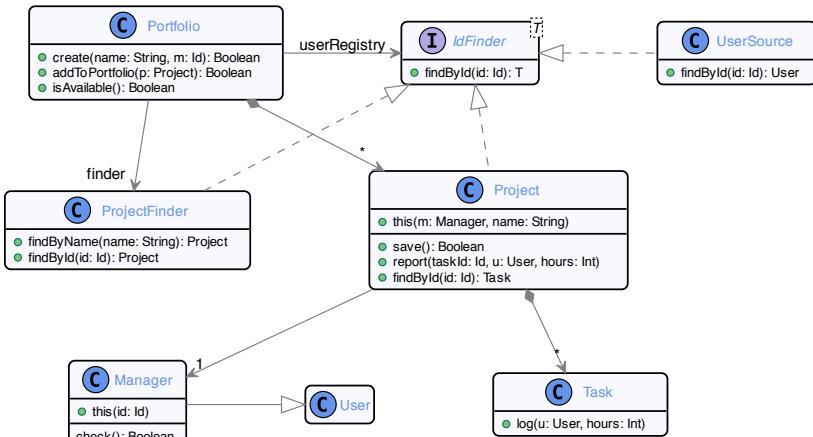
Fonctionnalité #3 : Rapporter du temps



81

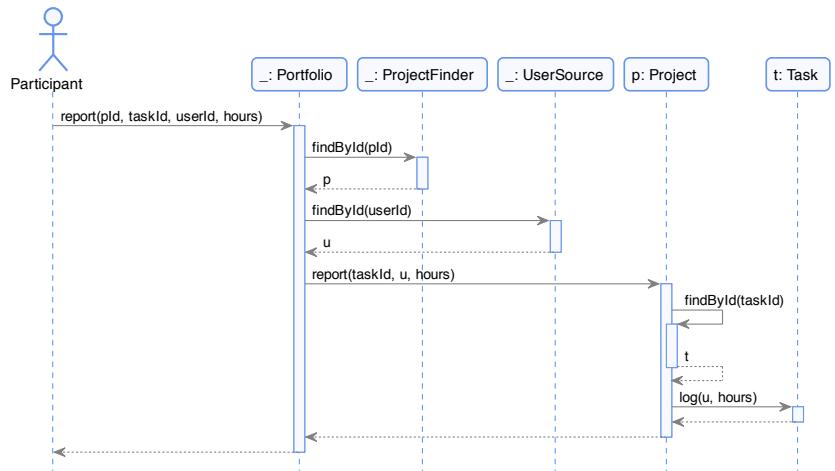
Il est temps de ré-usiner la conception

Fonctionnalité #3 : Rapporter du temps ✓



83

Fonctionnalité #3 : Rapporter du temps ✓



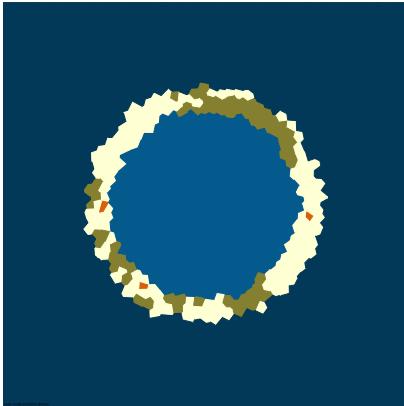
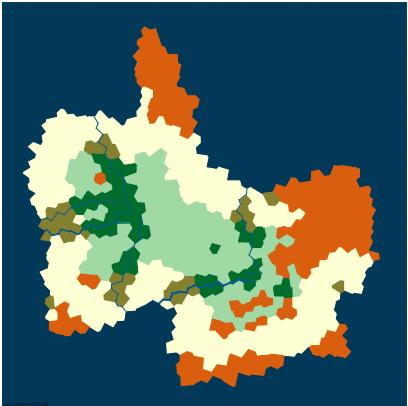
84

On pourrait faire beaucoup mieux

La conception est une activité qui demande du temps de repos

85

Objectif : Exploration d'îles inconnues



Générateur de terrain : Projet de recherche + Stages Maîtrise

87

Projet #2 Island

4

86

Génération procédurale de terrain

- Procédé classique en Jeu Vidéo
 - Initialement utilisé car les supports physique ne permettaient pas de stocker les niveaux par manque de mémoire !
 - Exemples : Diablo, instances dans les MMORPG, ...
- Principe :
 - On configure le générateur avec des exigences
 - Localisation géographique, nombre de rivières, ...
 - On lance la génération
- Chaque semaine, une île inconnue (**même par moi !**)

88

Principe d'exploration

- Vous pilotez **un drone** qui survole la **zone de recherche**
- L'avion est équipé :
 - *d'un moteur pour avancer*
 - *d'un gouvernail pour changer de cap*
 - *d'un sonar pour localiser la terre*
 - *d'un scanner pour analyser le terrain* survolé
- **Votre mission :**
 - Trouver les points d'interêt de l'île
 - Cartographier sa géographie

89

Condition de Victoire

- **Rentrer au port** et remplir les **objectifs** du commanditaire
 - Trouver l'île
 - Identifier les criques permettant le débarquement sur l'île
 - Identifier un site d'évacuation en cas de problème
 - Cartographier les biomes présents sur l'île
 - Estimer la surface de l'île
 - Estimer la quantité d'eau potable disponible
- **Si vous ne rentrez pas à temps, faites sortir le drone de la zone d'exploration, levez une erreur non traitée, c'est perdu.**

91

Protocole d'exploration

- La **simulation** fonctionne au **tour par tour**
- **Au premier tour** vous recevez votre **cap**, et votre "**budget**"
 - *Vous devez rentrer au port avant de l'avoir consommé (énergie)*
- **Pour chaque tour** :
 - Le moteur de simulation vous **demande votre décision**
 - *Il l'applique au terrain*
 - Il vous **envoie le résultat** de votre décision
- **Au dernier tour,**
 - vous devez **transmettre votre rapport de mission**

90

Championnat Hebdomadaire

- **Chaque mercredi**, votre projet sera mis en jeu sur une île
- Vous gagnez des points en **remplissant les objectifs**
- A la fin de la session, il y aura un **cadeau "symbolique"** pour le podium (top 3)
 - *P.-ex. un paquet de jujubes, des M&Ms, ...*
- **Le championnat n'est pas pris en compte dans l'évaluation**
 - *Il vous sert à mesurer votre progression et voir où vous vous positionnez dans la classe.*

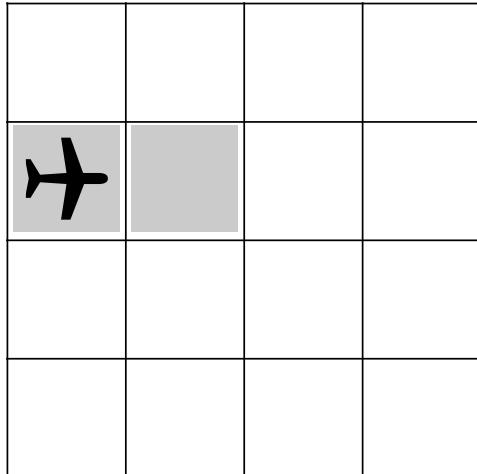
92

Mise en oeuvre concrète

```
public class Explorer implements IExplorerRaid {  
  
    @Override  
    public void initialize(String context) {  
        return;  
    }  
  
    @Override  
    public String takeDecision() {  
        return "{ \"action\": \"stop\" }";  
    }  
  
    @Override  
    public void acknowledgeResults(String results) {  
        return;  
    }  
  
    @Override  
    public String deliverFinalReport() {  
        return (new Report()).toString();  
    }  
  
}
```

93

Avancer (FLY)



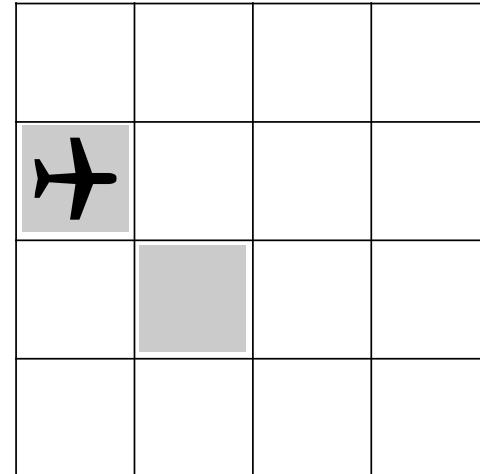
95

Décisions disponibles

- Liste des actions disponibles sur le drone :
 - Rentrer au port (**STOP**)
 - Avancer (**FLY**)
 - Changer de cap (**HEADING**)
 - Utiliser un des sonars (**ECHO**)
 - Scanner le terrain (**SCAN**)
- Chaque action consomme de l'énergie dans le budget
 - **Vous devez rentrer avant d'avoir épuisé la batterie**

94

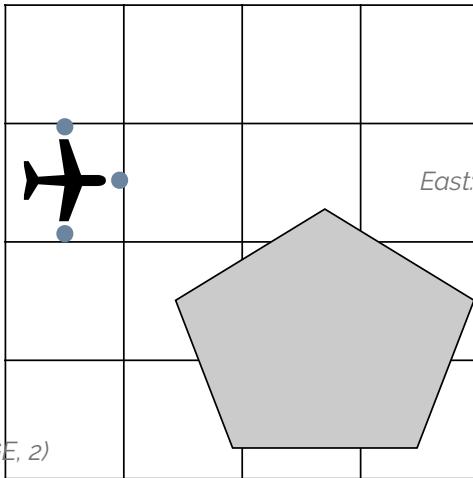
Changer de cap (HEADING)



96

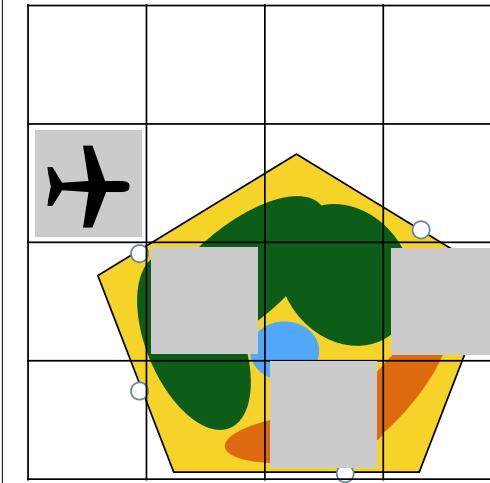
Utiliser le sonar (ECHO)

North: (OUT_OF_RANGE, 1)



Utiliser le scanner (SCAN)

SCAN
HEADING (SOUTH)
SCAN
HEADING (EAST)
SCAN
HEADING (NORTH)
SCAN



97

98

Pour la prochaine séance

Lisez l'étude de cas (Partie C) de l'examen intra H19

https://github.com/ace-lectures/A19-INF-5153/blob/master/docs/exams/19_H_1_intra.pdf

99

100