Kata

Junit / Mockito

Philippe Collet, d'après Sébastien Mosser et Simon Urli

Licence 3 MIAGE – S6

2015-2016

Université Nice SOPHIA ANTIPOLIS

Plan

- Game of Dices : specs
- Architecture (Maven)
- Tache 1 : Lancer un dé
- Tester (Junit)
- Mocker (Mockito)
- Tache 2 : Ajouter un joueur
- Tester
- Optional en Java 8
- Tache 3 : Garder le maximum de deux jets…
- Tache 4 : Jouer aux dés…
- Tache 5 : Refactoring de classes / nouveaux tests

Games of Dices : product backlog

- **Pas forcément de *stories* mais au moins un découpage vertical**

1. Etre capable de jeter un dé
   - Critère d'acception : le dé a 6 faces, et retourne un nombre aléatoire en 1 et 6.
2. Associer un jet de dé à un joueur précis
   - Critère d'acception : un joueur a un nom, et expose la valeur obtenue de son propre dé
3. Le joueur lance deux dés et garde le maximum
   - Critère d'acception : le dé est lancé juste 2 fois, et seulement la valeur maximale est conservée
4. Le jeu de dés se joue à 2, et le joueur qui obtient la valeur maximale après un lancer gagne (ex-aequo entraîne une relance, pas de vainqueur après 5 matches ex-aequo)
   - Critère d'acception : le jeu expose un vainqueur en suivant les règles

Architecture

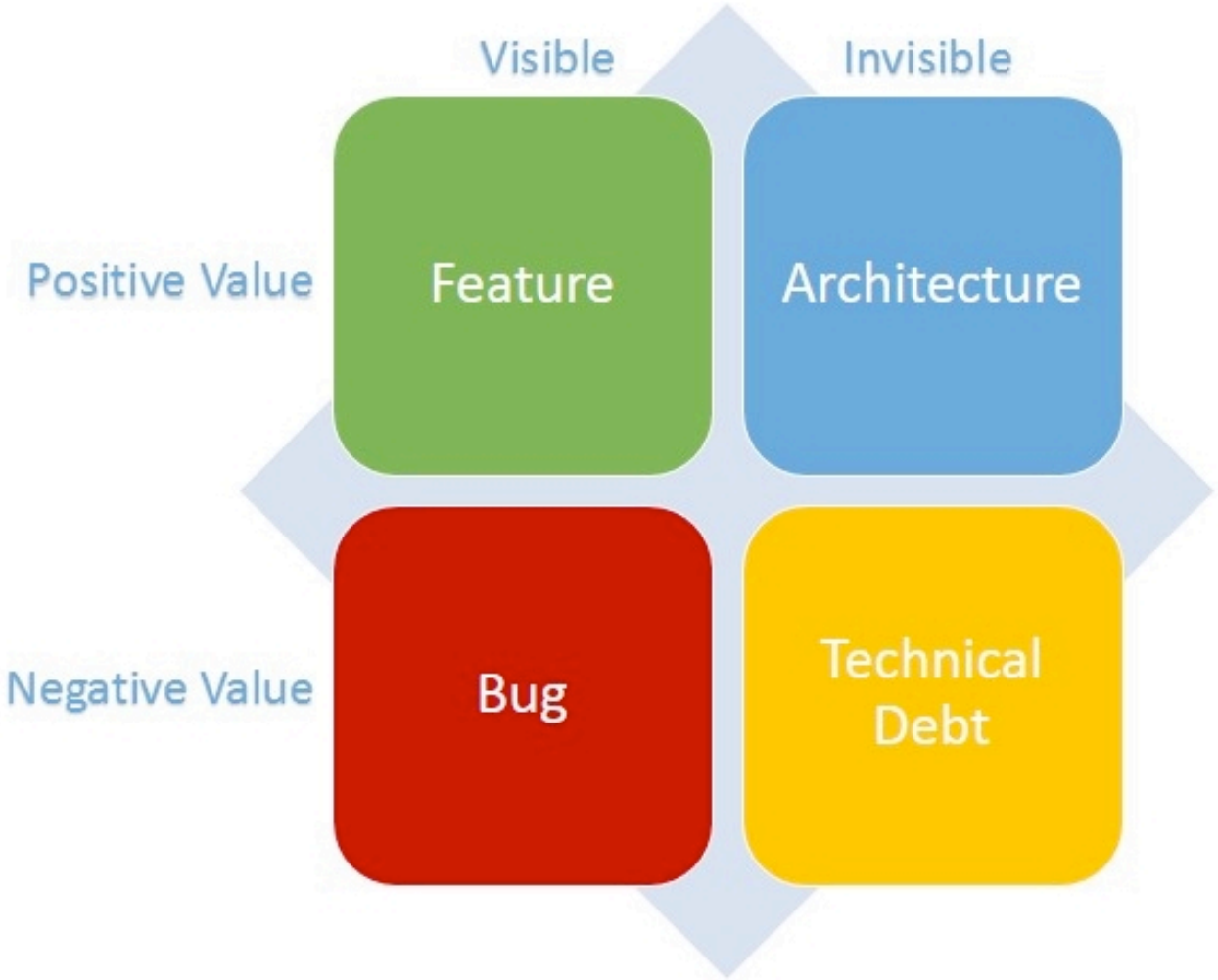
- Création des répertoires
  - Src, test
- Création d'un pom.xml minimal
- Pas forcément test-driven, mais testé unitairement et utilisation de mock

Tache 1 : Etre capable de jeter un dé

- Critère d'acception : le dé a 6 faces, et retourne un nombre aléatoire en 1 et 6.

- **Package god (Game Of Dices)**
- **Classe Dice**
  - **Méthode roll**
- **Reproductibilité du jeu ?**
  - **Objet random fourni à la création**
- **Si random ne marche pas bien ?**
  - **RuntimeException…**
  - **C'est clairement de la dette technique !**

Dette technique ?



Tester

- Modéliser les critères d'acceptation
  - Bonne propriété : quand les tests passent, la tâche est terminée !
- On a besoin de tester :
  - Lancer un dé rend une valeur ?
  - Une valeur hors de [1-6] lance une exception

Avec des mocks...

- Pas terrible de redéfinir une classe juste pour les tests
  - Si elle était plus compliquée, ce serait quasiment infaisable
- Placer mockito dans maven
- Créer un mock de Random

Tache 2 : Associer un jet de dé à un joueur précis

- Critère d'acception : un joueur a un nom, et expose la valeur obtenue de son propre dé

- **Classe Player**
  - **Constructeur avec le nom du joueur et le dé**
  - **Attribut lastValue (et getter) pour la dernière valeur du dé (-1 sinon)**
  - **Méthode play lance le dé et stocke la valeur**

- **Test ?**

Optionals en Java 8

- Renvoyer -1, c'est moche -> Dette technique encore…
- Optional : encapsuler un objet qui peut être *null*
- On se met en Java 8
  - On s'en assure dans le pom.xml
- Modifier la classe Player
  - lastValue est un Optional
- Modifier les tests
  - Vérifier juste que la valeur est présente ou pas…

Tache 3 : Le joueur lance deux dés et garde le maximum

- Critère d'acception : le dé est lancé juste 2 fois, et seulement la valeur maximale est conservée

- **Modifier la méthode play pour lancer deux dés et garder le max**

- **Tests ?**
  - **Le joueur suit bien les règles**
  - **Le joueur garde la valeur max**

Tache 4 : Le jeu de dés se joue à 2...

- et le joueur qui obtient la valeur maximale après un lancer gagne (ex-aequo entraîne une relance, pas de vainqueur après 5 matches ex-aequo)

- Critère d'acception : le jeu expose un vainqueur en suivant les règles

- **Création d'une classe Game**
  - **Deux joueurs (left, right)**

- **Tests ?**
  - **Cas « pas de vainqueur »**
  - **Cas « un vainqueur »**

Tache 5 :

- La classe Game « voit » un résultat entier provenant du Dice
  - Pas très abstrait
  - Deux niveaux d'indirection -> couplage fort

- **Création d'une classe PlayResult**
  - **Qui permet la comparaison des résultats**

- **Refactoring des classes Game et Player**

- **Tests ?**