

# **LOG210 ANALYSE ET CONCEPTION DE LOGICIELS: SÉANCE 08**

# SURVOL

- Travail en équipe
- Rappel méthodologie
- Rétroaction mini-test
- Passer des diagrammes au code
- Dette technique (7.x) et Réusinage (ndc 11.x)
- Architecture logique (en couches) F12.1-12.8/A13.1-13.8
- Conception de packages (ndc 18)



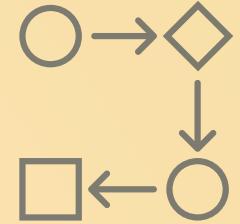
# TRAVAIL EN ÉQUIPE

## DÉVELOPPEMENT DE LOGICIELS



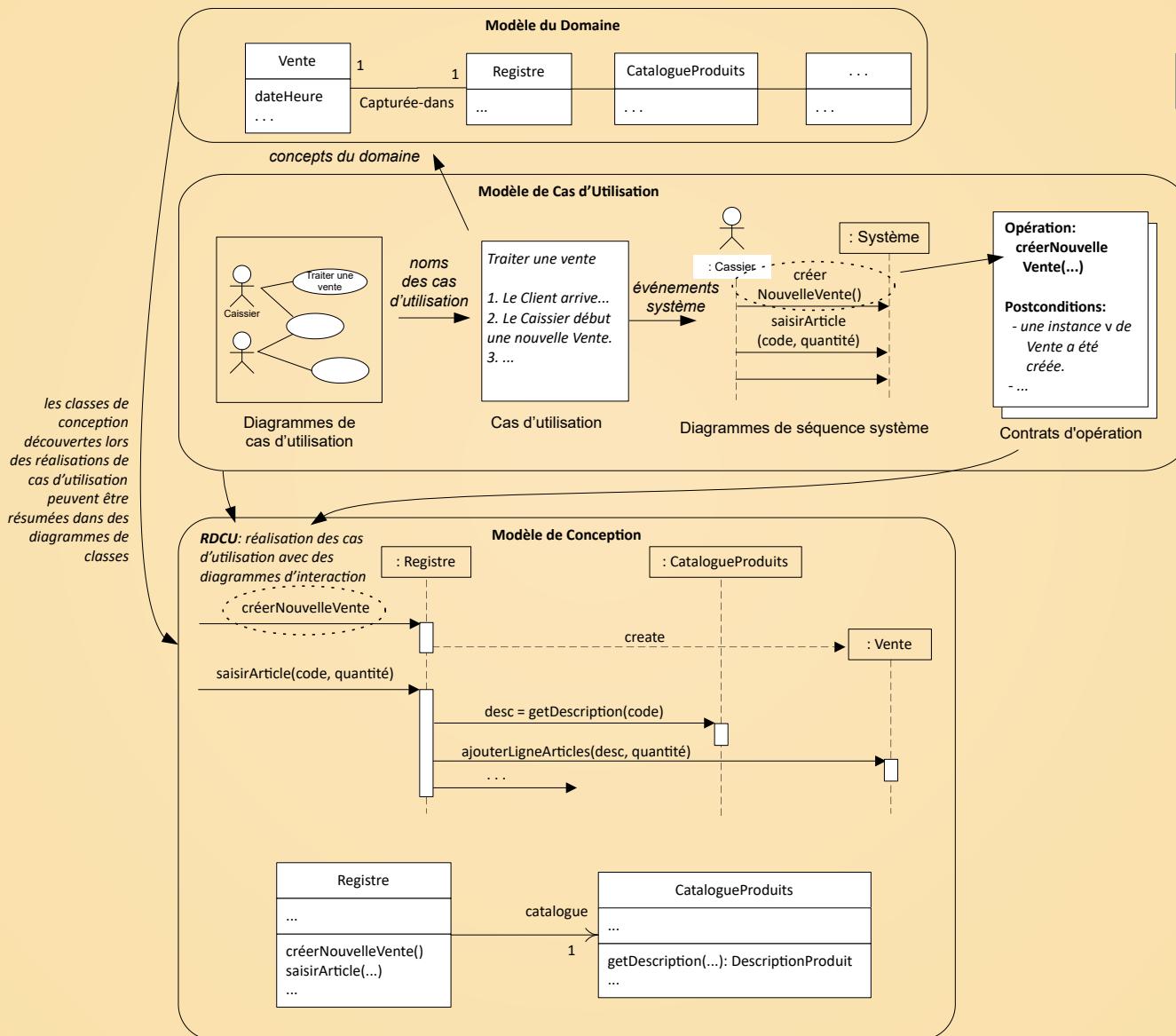
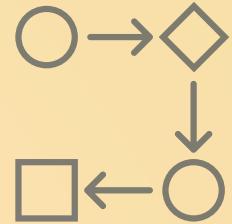
# TRAVAIL D'ÉQUIPE

- Quels sont les problèmes HRC rencontrés par votre équipe?
- Quelles solutions avez-vous apportées?



# MÉTHODOLOGIE

# SURVOL



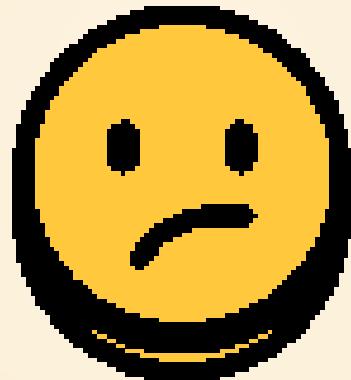


# RÉTROACTION

## MINI-TEST



# QUESTIONS DIFFICILES



Selon les statistiques de hier matin.

# ARCHITECTURES EN COUCHES



Socrative → ETSLOG210

Dans quelle couche trouve-t-on des règles pour la vente des produits dans une application telle que NextGen, ou pour un jeu comme Monopoly ?

- A. Présentation
- B. Application
- C. Domaine(s)
- D. Infrastructure métier
- E. Services techniques
- F. Fondation

# ARCHITECTURES EN COUCHES



Socrative → ETSLOG210

Dans quelle couche trouve-t-on un service de reconnaissance de la parole (comme Siri, Alexa, etc.) ?

- A. Présentation
- B. Application
- C. Domaine(s)
- D. Infrastructure métier
- E. Services techniques
- F. Fondation

# ARCHITECTURES EN COUCHES



Socrative → ETSLOG210

Dans quelle couche trouve-t-on un service pour calculer l'arc tangente d'un nombre réel ?

- A. Présentation
- B. Application
- C. Domaine(s)
- D. Infrastructure métier
- E. Services techniques
- F. Fondation

# DETTE TECHNIQUE



Socrative → ETSLOG210

Le hacking cowboy amène de la dette technique...

- A. parce qu'on écrit vite du code qui fonctionne, sans porter une attention à sa lisibilité, à son extensibilité, etc.
- B. parce que le coût de la main-d'œuvre en programmation augmente avec le temps.
- C. parce que le code écrit rapidement a plus de bogues à corriger.

# RÉUSINAGE



Socrative → ETSLOG210

Qu'est-ce qui n'est **pas** un but ou une activité de la refactorisation?

- A. supprimer le code dupliqué
- B. rendre le code plus facile à lire et à comprendre
- C. faire en sorte que les méthodes soient plus courtes
- D. éliminer l'utilisation de constantes littérales codées en dur
- E. supprimer les commentaires en rendant le code plus descriptif
- F. rendre le code plus rapide
- G. corriger les bogues dans le code



# RÉUSINAGE



Socrative → ETSLOG210

Un “code smell” concerne quelle qualité du logiciel?

- A. Fonctionnalité
- B. Facilité d'utilisation (*usability*)
- C. Fiabilité (*reliability*)
- D. Performance
- E. Possibilités de prise en charge (*supportability*)

# RÉUSINAGE



Appariez l'activité de réusinage (refactoring) avec son principe GRASP

- A. Remplacer des codes (switch, if-else) d'une logique sur les types avec des sous-classes avec une implémentation différente pour chaque méthode.
- B. Extraire des méthodes d'une classe pour former une nouvelle classe.
- C. Renommer une classe.
- D. Renommer une variable.
- E. Introduire objet de paramètre.
- F. Déplacer une fonction.
- G. Extraire une fonction.

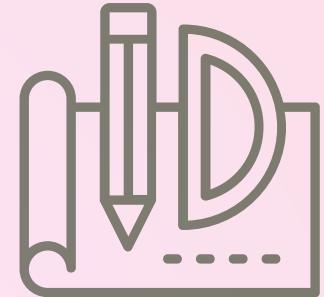
# CONCEPTION DE PACKAGES



Socrative → ETSLOG210

Quels sont les principes d'organisation de packages présentés dans les lectures?

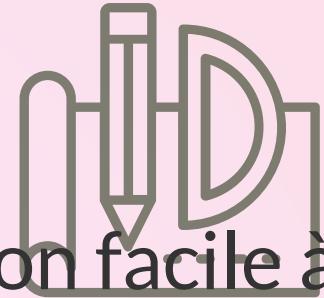
- A. Pas de couplage dans les packages
- B. Organiser les packages en partitions verticales et horizontales fonctionnellement cohésives.
- C. Packager une famille d'interfaces.
- D. Pas de cycles dans les packages.
- E. Créer un package par tâche et par groupe de classes instables.
- F. Factoriser les types indépendants.
- G. Pas de polymorphisme dans les packages
- H. Utiliser les Fabrications pour limiter la dépendance aux packages concrets.
- I. Packager les classes par cas d'utilisation



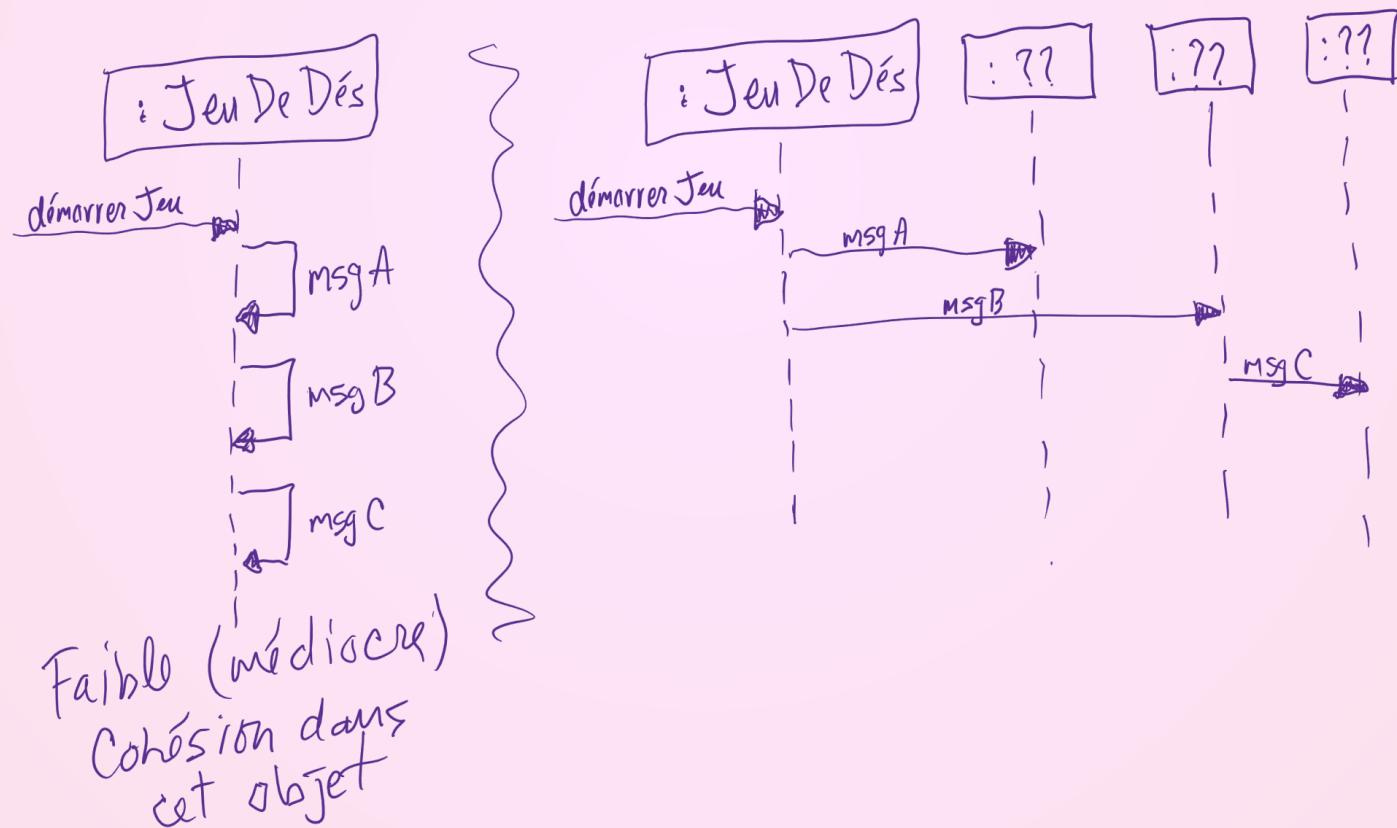
# RDCU

## RÉALISATION D'UN CAS D'UTILISATION

# RDCU

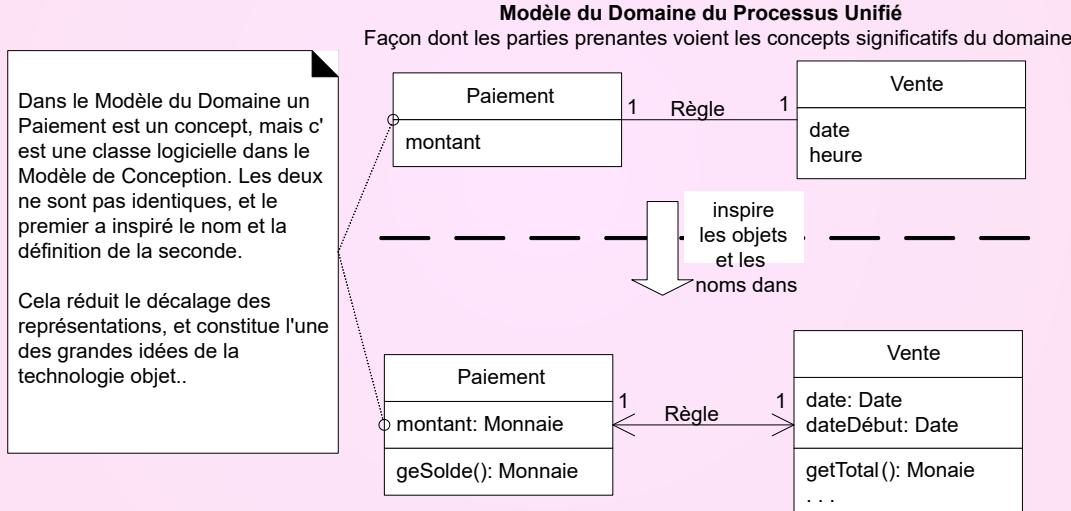


Prendre les bonnes décisions pour une solution facile à comprendre et modulaire...



# DÉCALAGE DES REPRÉSENTATIONS

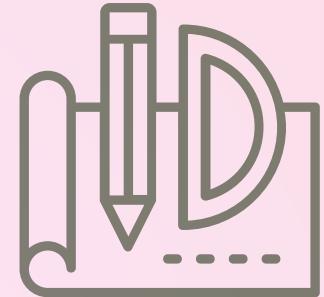
Facile? Les classes logicielles devraient ressembler à des classes conceptuelles (concepts du monde réel).



**Modèle de Conception du Processus Unifié**  
Le développeur orienté objet s'est inspiré du domaine  
du monde réel pour créer les classes logicielles.

En conséquence, le décalage des représentations entre la façon dont les parties prenantes voient le domaine et sa traduction logicielle a été réduit.

## Qui fait quoi? Qui a quelle responsabilité?



# RDCU

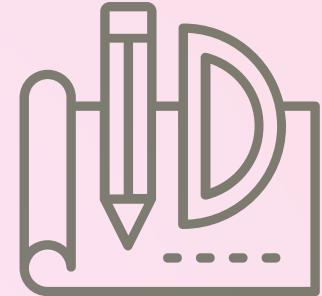
Approche: conception orientée-responsabilités

## GRASP

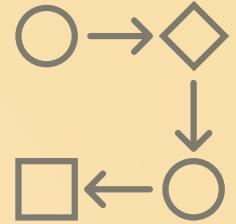
General Responsibility Assignment Software Patterns

Pour décider où mettre les méthodes...

# GRASP



- Contrôleur (séparation des couches)
- Createur
- Expert en information
- Faible couplage
- Forte cohésion
- Polymorphisme
- Fabrication pure
- Indirection
- Protection de variation

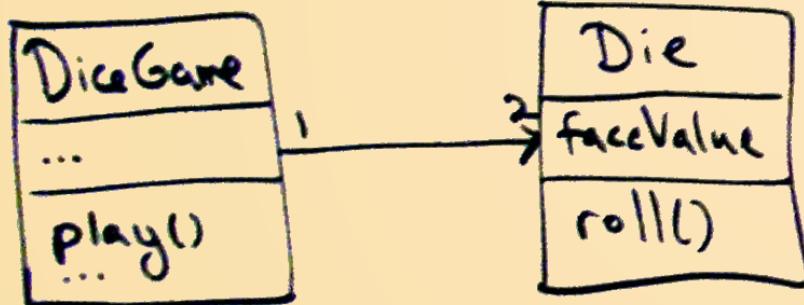


# PASSER DES DIAGRAMMES AU CODE

Ne pas oublier la **Visibilité** des objets (ndc 9.3)

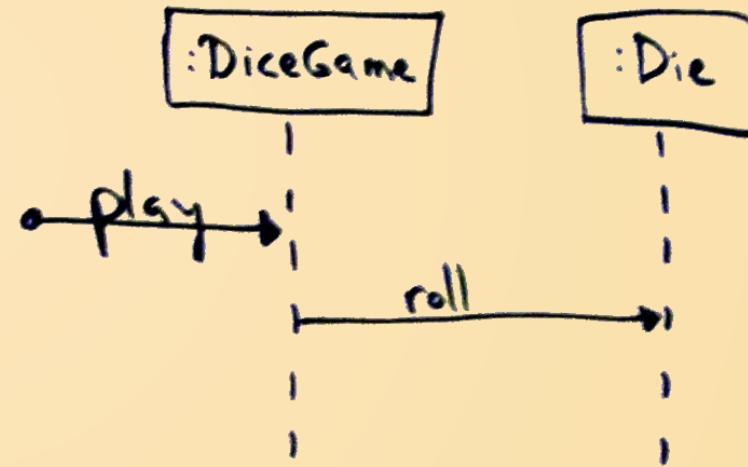
# QU'EST-CE QUI EST PLUS IMPORTANT?

Static model



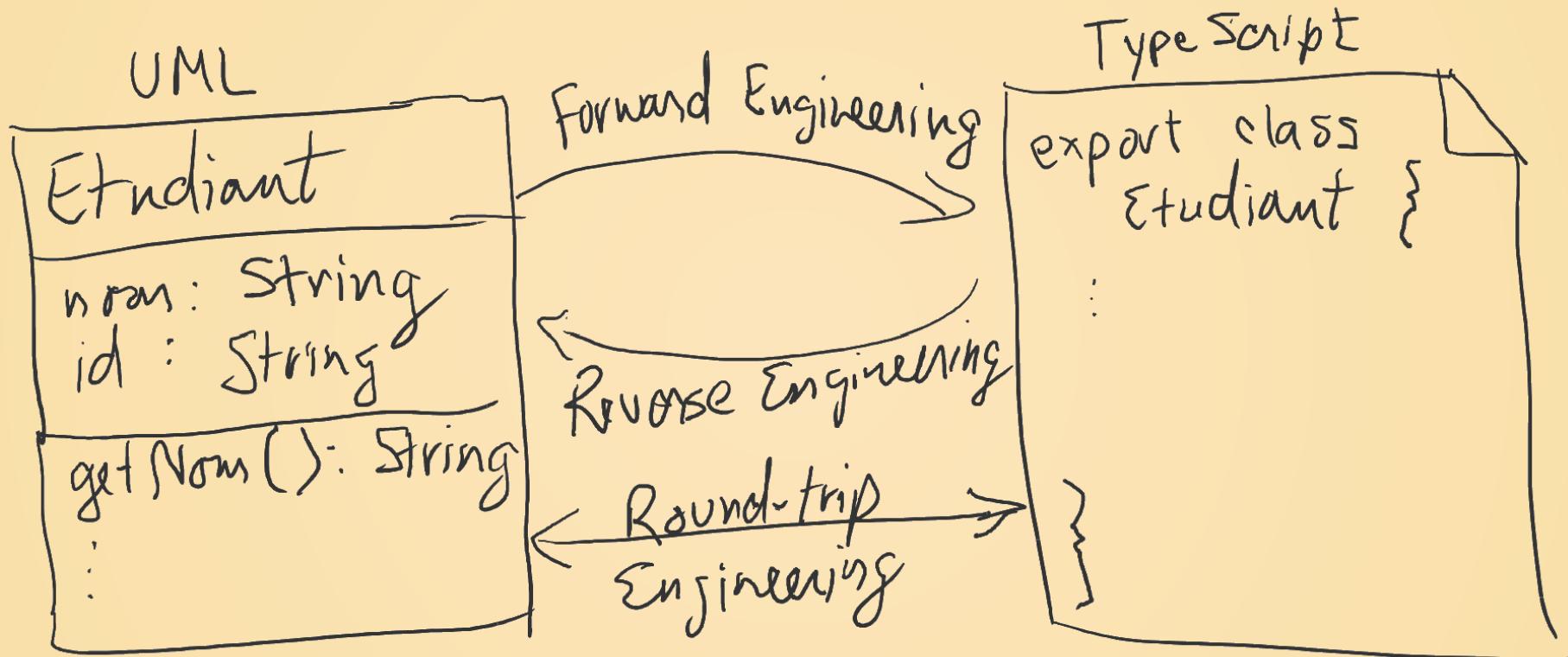
UML Class Diagram

Dynamic Model



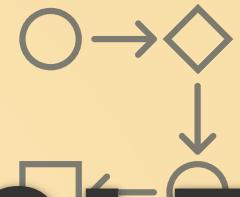
UML Sequence Diagram

# OUTILS POUR AUTOMATISER LE PASSAGE



# **EXERCICE SUR GOOGLE CLASSROOM**

LOG210: Créer des méthodes à partir des diagrammes  
d'interaction



# **DETTE TECHNIQUE (NDC 7.X) ET RÉUSINAGE (NDC 11.X)**

# **EXEMPLES CONCRETS DE DETTE TECHNIQUE**

# CODE SMELLS

- Nom mystérieux
- Code dupliqué
- Longue fonction
- Données globales
- Grosse classe
- ...

# EXEMPLES DES LABORATOIRES



Patiner (Québec): Éluder une question embarrassante.

# NOM MYSTÉRIEUX



## Orthographe

```
export class Question {  
    // classe inspirée de la classe conceptuelle (du MDD)  
    private _type: string;  
    private _tag: Array<String>;  
    private _ennonce: string;  
    ...
```

```
public getEtudiantParEmail(email: string): Etudiant { ... }
```



Solution: renommer

# NOM MYSTÉRIEUX



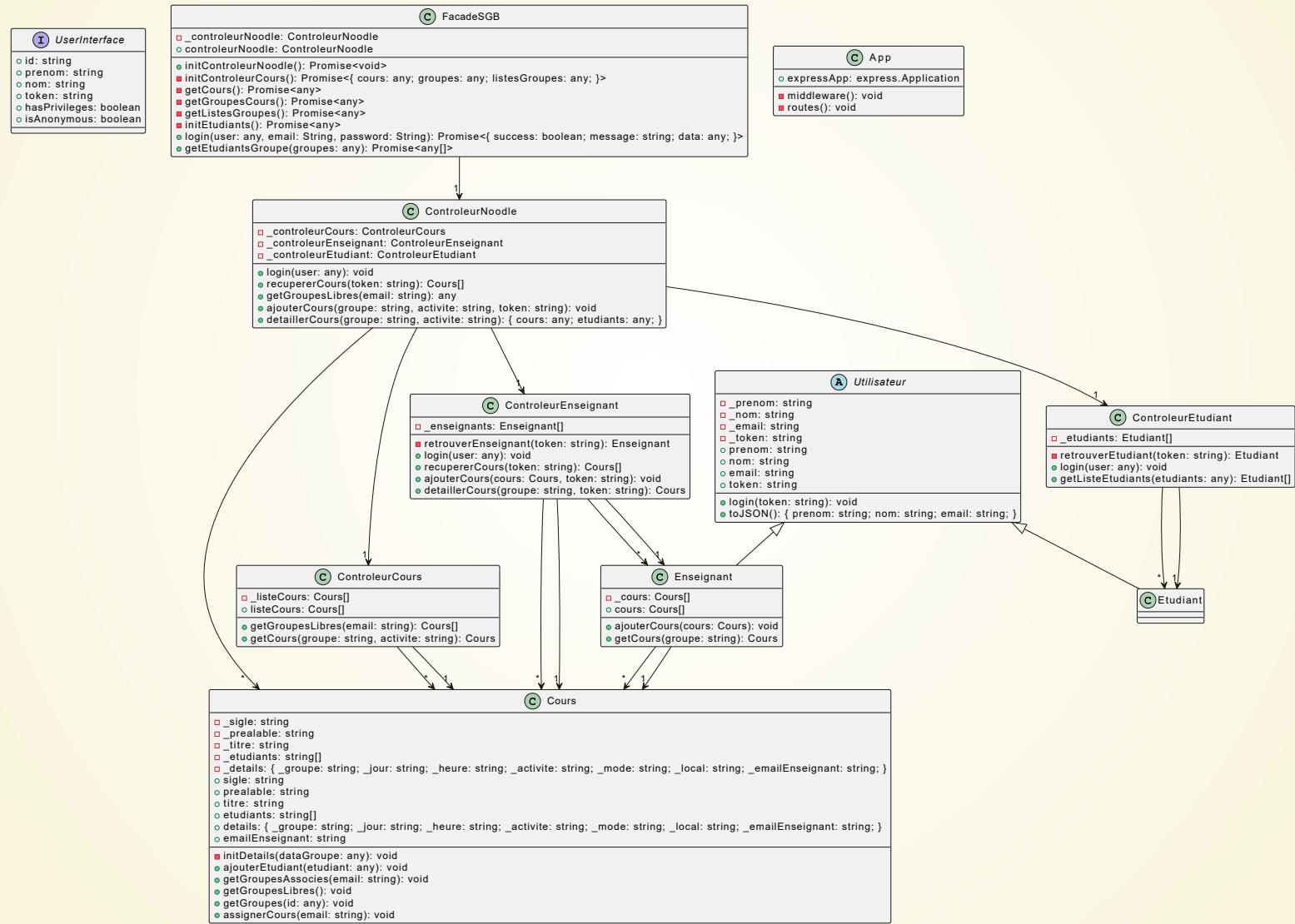
Méthode sans action claire (verbe)

```
public listeQuestions(...) { }  
  
public nouvelleQuestion(...) { }  
  
private _errorCode500(...) { }
```



Solution: Renommer

# NOM(S) MYSTÉRIEUX (COHÉRENCE)



# NOM(S) MYSTÉRIEUX



- UserInterface vs. classe abstraite Utilisateur
- attributs dupliqués (FacadeSGB, Utilisateur, ControleurCours, Cours)

```
_controleurNoodle: ControleurNoodle  
controleurNoodle: ControleurNoodle
```

- login() dans 4 classes (sans polymorphisme)
- FaçadeSGB dépend des contrôleurs plutôt que l'inverse

 Solution: Renommer selon une norme, supprimer le code inutile ( sans les tests, c'est risqué!)

# CODE DUPLIQUÉ

```
.messages
  if (messages.info)
    each message in messages.info
      div.flash.alert.alert-info.col-sm-6 #{message}
  if (messages.win)
    each message in messages.win
      div.flash.alert.alert-success.col-sm-6 #{message}
  if (messages.error)
    each message in messages.error
      div.flash.alert.alert-danger.col-sm-6 #{message}
```

Dans les fichiers ajouterCours.pug,  
ajouterQuestion.pug, listeQuestion.pug, ...

# CODE DUPLIQUÉ (PUG)



Solution: Mixins

<https://pugjs.org/language/mixins.html>

# LONGUE FONCTION

50+ lignes

```
public async login(...) {
    const responseData = { ... }

    // En premier, on tente d'effectuer la connexion avec les ens
    try { ... } catch (e) { ... }

    // Si la connexion comme enseignant échoue, on tente d'effect
    if (!responseData.success) {
        try { ... } catch (e) {
            // Si la connexion échoue, les identifiants ne sont p
            ...
        }
    }

    if (responseData.success) { ... }
```

# LONGUE FONCTION (SOLUTION)

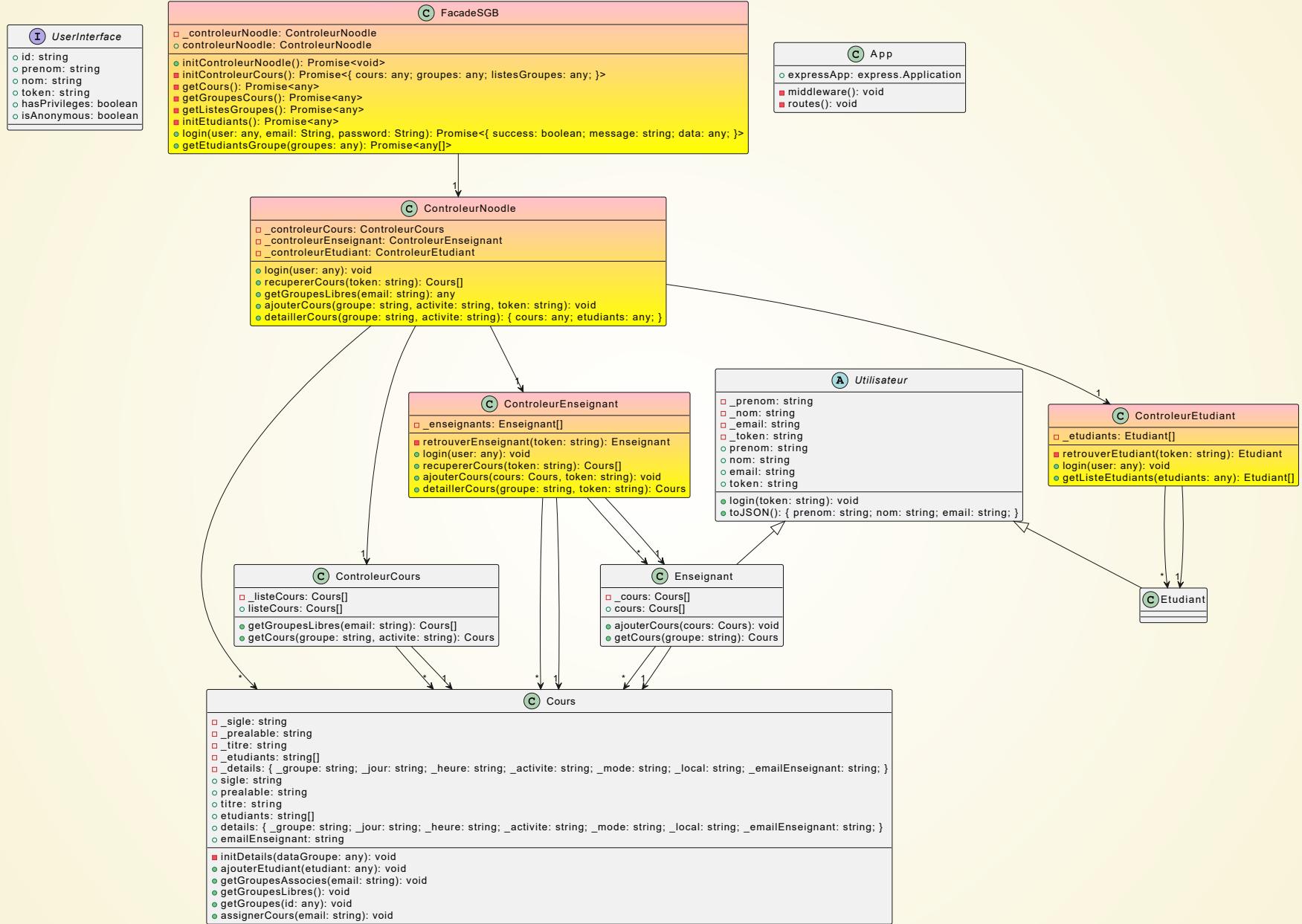
- Solution: Réusinage “extract function”
- Les commentaires sont des indices et deviennent obsolètes

```
// En premier, on tente d'effectuer la connexion avec les enseignants
try {...} catch (e) {...}
```

 Extraire fonction avec nom qui ressemble au sens du commentaire:

```
const resultat = essayerLoginEnseignants(...)
```

# GROSSES CLASSES



# GROSSES CLASSES

Qu'arrive-t-il lorsqu'on veut supporter les fonctionnalités de Devoir, Quiz, etc.?

Ces contrôleurs vont se trouver avec toutes ces (nouvelles) responsabilités.

# GROSSES CLASSES



Solution

- appliquer GRASP Contrôleur (2<sup>e</sup> choix):  
GestionnaireX, où X est le nom du cas d'utilisation
- GestionnaireCréerCours,  
GestionnaireRemettreDevoir, etc.

# D'AUTRES EXEMPLES

Blogue de CMU: « Managing the Consequences of Technical Debt: 5 Stories from the Field »

- À la va-vite (Quick-and-Dirty) if-then-else
- Hitting the Wall
- Crumbling Under the Load
- Death by a Thousand Cuts
- Tactical Investment

# À LA VA-VITE IF-THEN-ELSE

Compagnie canadienne (anglaise) qui a voulu vendre leur logiciel au Québec. Une semaine de hacking cowboy  pour « franciser » leur logiciel:

```
// variable globale
#define French = "yes" // "no"

if (French == "yes") {
    ...
} else {

}
```

Vendeur va au Japon et déclare le logiciel « internationalisé » 

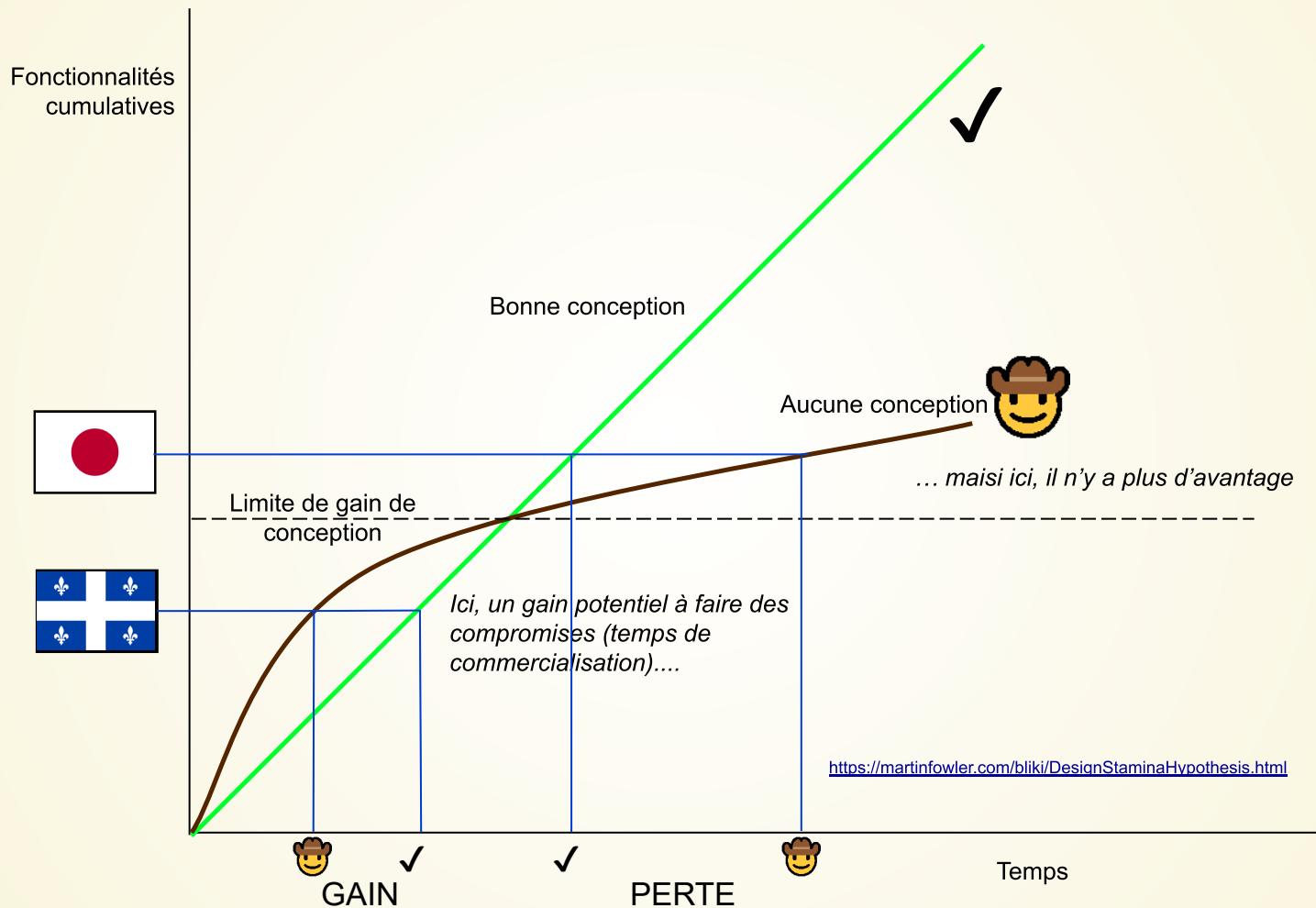
# À LA VA-VITE IF-THEN-ELSE

Internationaliser « correctement » pour le français (et d'autres langues) aurait pris plus de temps et retardé la vente du produit au Québec.

Le hacking cowboy a permis une vente plus rapide, mais a créé un cauchemar pour rendre le code utilisable au Japon.

Cette « dette technique » a permis d'aller plus vite pour le Québec, mais va retarder la livraison pour le Japon.

# À LA VA-VITE IF-THEN-ELSE



# PENTE DU TUNNEL ENTRE LE A ET LE B



Exemple de « solution » à un problème de mauvaise information (incomplète). On découvre des conduits qui n'étaient pas sur les plans...

Creuser plus bas sur les deux côtés pour faire un tunnel à niveau coûterait trop cher. La solution n'est pas idéale, mais acceptable.



Socrative → ETSLOG210

Qu'est-ce qui n'est pas un exemple de dette technique?

- A. Il faut que ça marche en 48 heures, alors pas revue de conception ou de code.
- B. On a raté un peu l'implémentation ici, mais la démo a convaincu un nouveau client.
- C. Nos développeurs coûtent cher, mais ça vaut l'investissement.
- D. Les singletons partout, ça ne fait pas de mal.
- E. L'importateur XML est robuste, mais difficile à comprendre. On croit que le format ne changera pas.



Socrative → ETSLOG210

Une conséquence souhaitée d'un réusinage  
(``refactoring``) est que le code s'exécute plus  
rapidement.

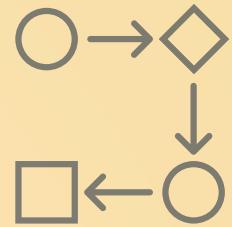
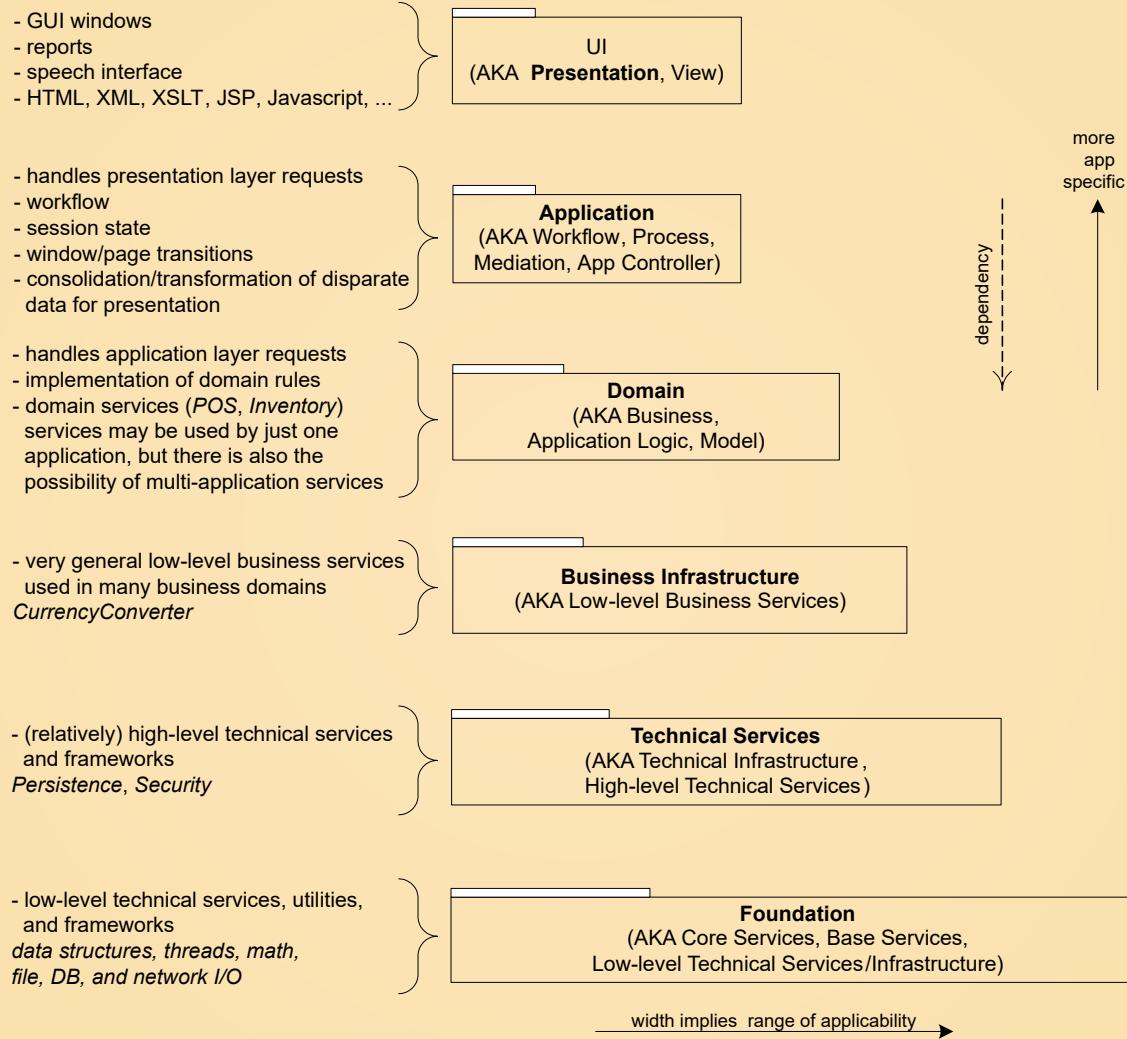
**Vrai ou Faux?**

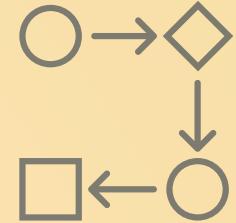


# **ARCHITECTURE LOGIQUE (EN COUCHES)**

**F12.1-12.8/A13.1-13.8**

# FIG. A13.4 ARCHITECTURE LOGIQUE

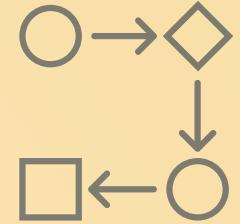




Socrative → ETSLOG210

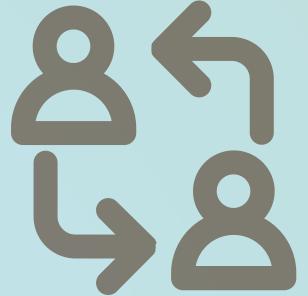
Quel concept de LOG121 est le plus proche de la notion d'architecture logique (en couches)?

- A. Polymorphisme
- B. Patron itérateur
- C. MVC
- D. Compilateur Java
- E. Patron méthode fabrique



# CONCEPTION DE PACKAGES

Notes du cours. Chapitre 18.



Created by Prithvi  
from the Noun Project

# FEUILLE D'UNE MINUTE

SVP m'écrire un courriel pour dire ce qu'étaient les points les moins clairs de la séance.