

**GÉNIE LOGICIEL ORIENTÉ OBJET (GLO-2004)**  
**ANALYSE ET CONCEPTION DES SYSTÈMES ORIENTÉS OBJETS (IFT-2007)**

**Automne 2016**

**Module 12 - Grands principes en conception orientée objet  
(partie 2)**

**Martin.Savoie@ift.ulaval.ca**

Bachelier Génie logiciel, Chargé de cours,  
département d'informatique et de génie logiciel

# Grands principes (GRASP)

1. Créateur
2. Expert en information
3. Contrôleur
4. **Faible couplage**
5. **Forte cohésion**
6. Polymorphisme
7. Fabrication pure
8. Indirection
9. **Protection contre les variations**

(modularité)

Caractéristiques  
recherchées

Recettes

On les trouve au  
chapitre 22 (version  
française) ou au  
chapitre 25 (version  
anglaise)

# Modularité (GRASP)

- Nous avons étudié au module précédent deux grands principes qui favorisent la modularité:
  - le *faible couplage*; et
  - la *forte cohésion*.
- Nous ajoutons maintenant le grand principe la *protection contre les variations*.

# Principe 9 : Protection contre les variations

- **Problème:**

- Comment concevoir des objets, des sous-systèmes ou des systèmes de telle façon que les variations ou l'instabilité de ces éléments n'aient pas d'impact indésirable sur d'autres éléments?

- **Solution:**

- Prévoir les éléments qui risquent fort de changer
- Les encapsuler par une interface stable (cacher le design interne d'éléments qui risquent de changer grâce à une interface stable)

# Protection contre les variations

- Plusieurs autres grands principes en sont des applications directes
- Comme par exemple:
  - Contrôleur!

# Nouveaux grands principes (GRASP)

## Partie 2

1. Créateur
2. Expert en information
3. Contrôleur
4. Faible couplage
5. Forte cohésion

(modularité)

Caractéristiques  
recherchées

- 6. Polymorphisme**
- 7. Fabrication pure**
- 8. Indirection**
- 9. Protection contre les variations**

Recettes

On les trouve au  
chapitre 22 (version  
française) ou au  
chapitre 25 (version  
anglaise)

# Principe 7 : Fabrication pure

- **Problème:**

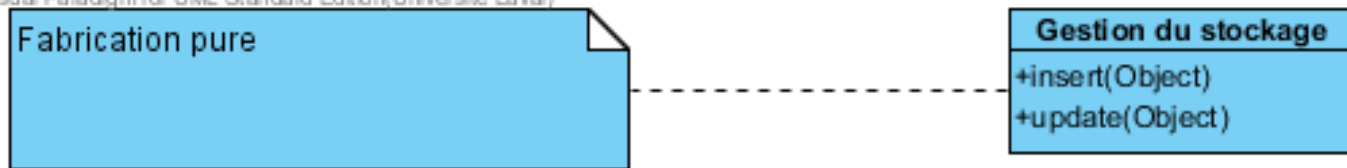
- Si les solutions recommandées par les grands principes semblent toujours mener à des designs avec trop de couplage et/ou manque de cohésion

- **Solution:**

- Affecter un ensemble de responsabilités fortement cohésif à une classe de commodité « artificielle » qui ne représente pas un concept du domaine.
- C'est une **pure** fabrication de l'**imagination**, afin de permettre un design **pur**.

# Exemple classique: sauvegarder des objets dans une base de données (rappel)

Visual Paradigm for UML Standard Edition (Université Laval)





# Et dans votre projet de session?

- Sous-système d'affichage: un incontournable

# Utilisation d'un JPanel pour l'affichage

- Voir livre « Java for Programmers » (Deitel)
- <http://proquest.safaribooksonline.com/?uiCode=univlaval&xmlId=9780137018529>
- Section 11.15 : exemple avec un panel

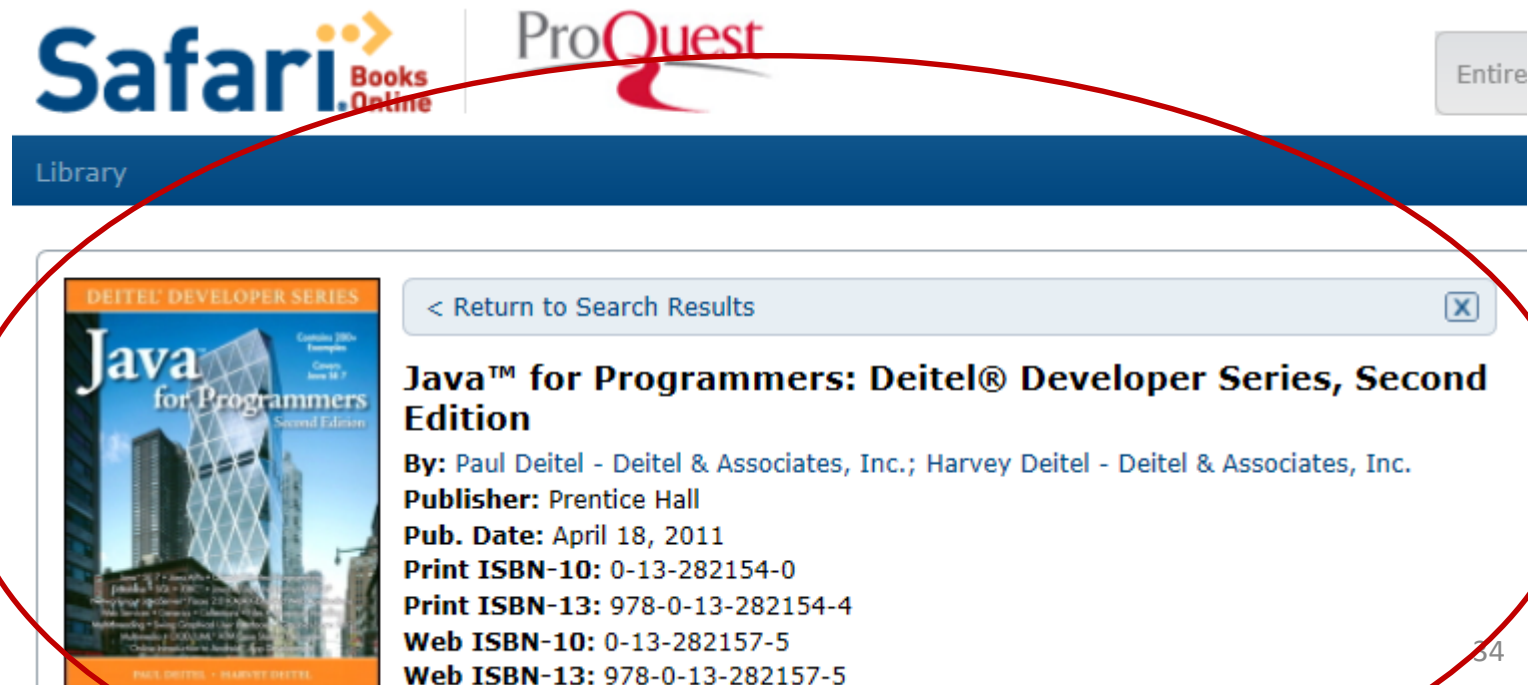
```
• 39  public void paintComponent( Graphics g )
• 40  {
• 41      super.paintComponent(g ); // clears drawing area
• 44      afficheurDeReseau.affiche(g, plan, ...)
• 46  }
```

On doit définir cette méthode qui sera appelée automatiquement par "Java" lorsque ce sera le temps de repeindre le contenu du panel (on n'appelle pas nous-même cette méthode)

On ne dessine pas directement le plan dans cette méthode. On redirige l'appel vers notre afficheur de plan, pour une plus grande réutilisabilité?

# Livres de Java

- Disponible en version électronique (service « Safari » de la bibliothèque)
- <http://proquest.safaribooksonline.com/?uiCode=univlaval&xmlId=9780137018529>



**Safari** Books Online | ProQuest

Library

Entire

< Return to Search Results

**Java™ for Programmers: Deitel® Developer Series, Second Edition**

**By:** Paul Deitel - Deitel & Associates, Inc.; Harvey Deitel - Deitel & Associates, Inc.

**Publisher:** Prentice Hall

**Pub. Date:** April 18, 2011

**Print ISBN-10:** 0-13-282154-0

**Print ISBN-13:** 978-0-13-282154-4

**Web ISBN-10:** 0-13-282157-5

**Web ISBN-13:** 978-0-13-282157-5

# Ressources supplémentaires (contrôleur et affichage)

- Wiki (ateliers 2 et 3) (Panel, Affichage, Contrôleur)
- Livre Java Chapitre 11 (notamment sec. 11.15)
  - <http://proquest.safaribooksonline.com/?uiCode=univlaval&xmlId=9780137018529>
- Larman: 17.13 (anglais) ou 16.13 (français)

# Principe 6 : Polymorphisme

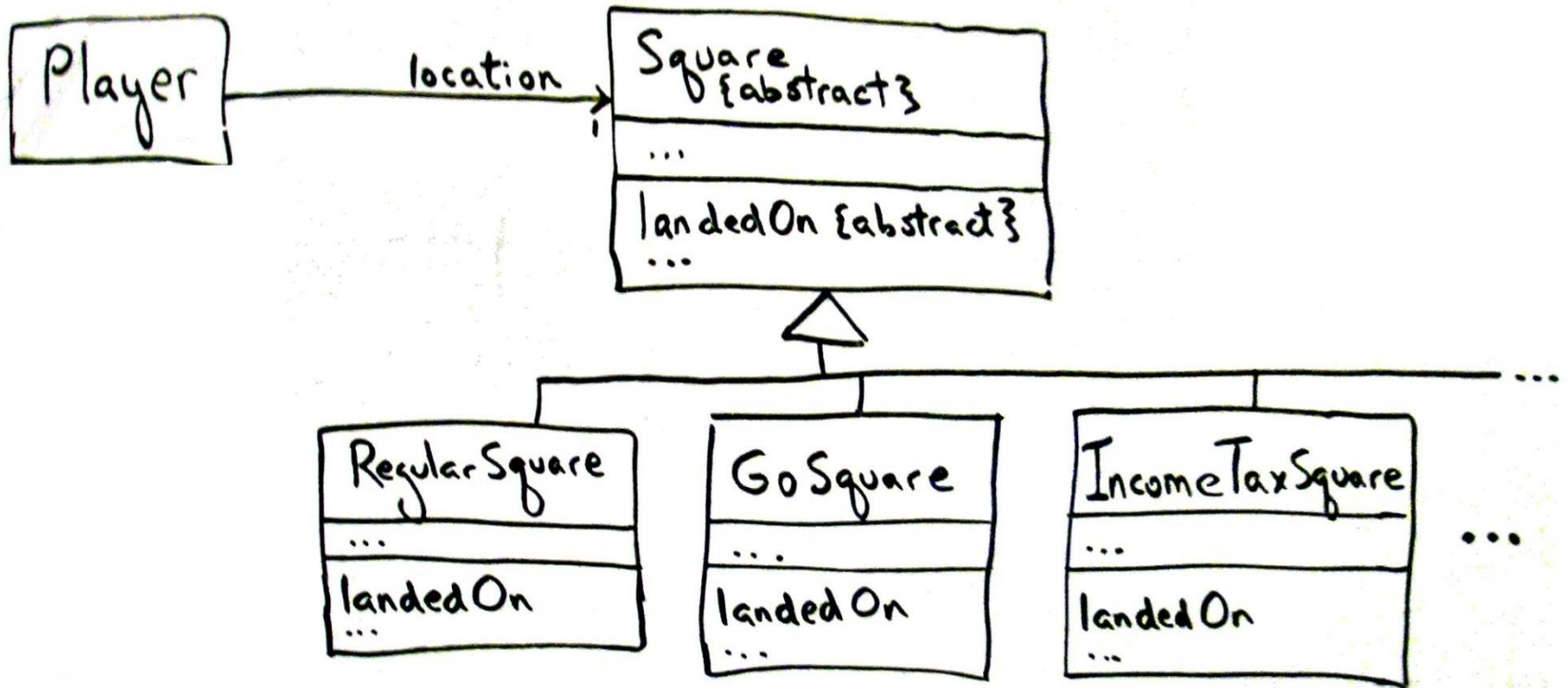
- **Problème:**

- Comment gérer des alternatives dépendantes du type de la classe ?
- Comment gérer des composants logiciels « enfichables » (plugins) ?

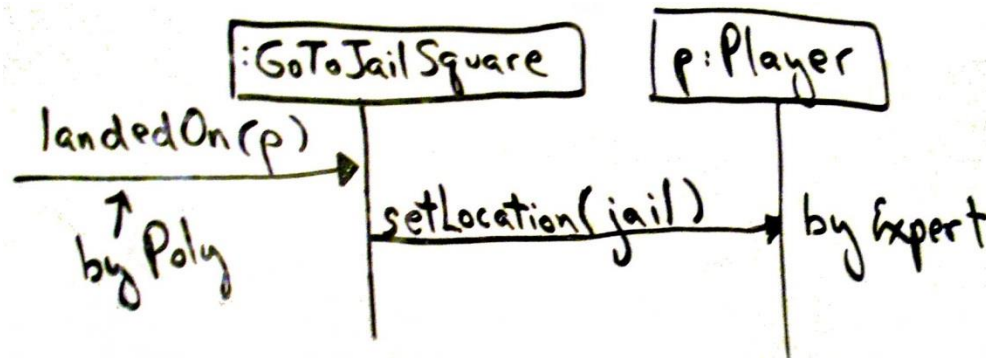
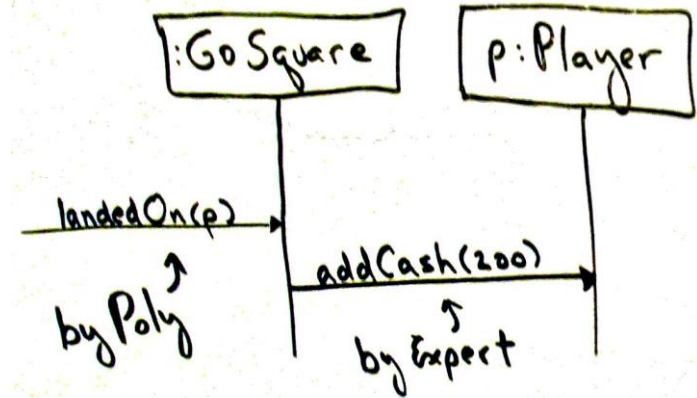
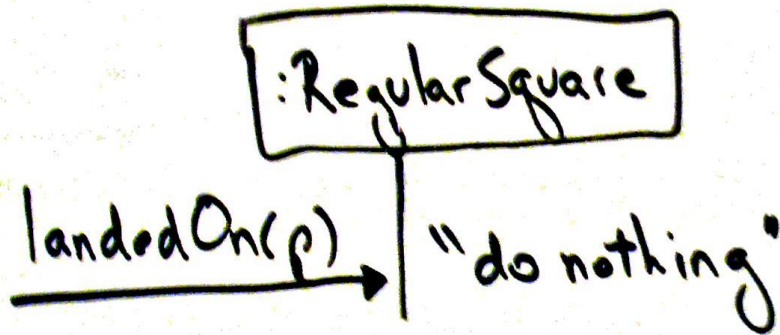
- **Solution:**

- Quand des fonctions ou des comportements connexes varient en fonction du type de classe, utiliser des classes / méthodes polymorphes

# Monopoly



# Monopoly



# Alternative que l'on cherche à éviter

- if then else if then else if then else
- switch case case case
- Pourquoi?
  - Le code appelant devrait alors « connaître » toutes les classes implantant les différentes variantes (amène un couplage trop fort)
  - Si on ajoute une nouvelle option, on doit modifier le code qui gère ces options (bouts de code qui fait les « if »)



# Exemple: Robo sapiens

[Ma\_procedure]

Si interrupteur 1 actif, alors moteur 1 positif,

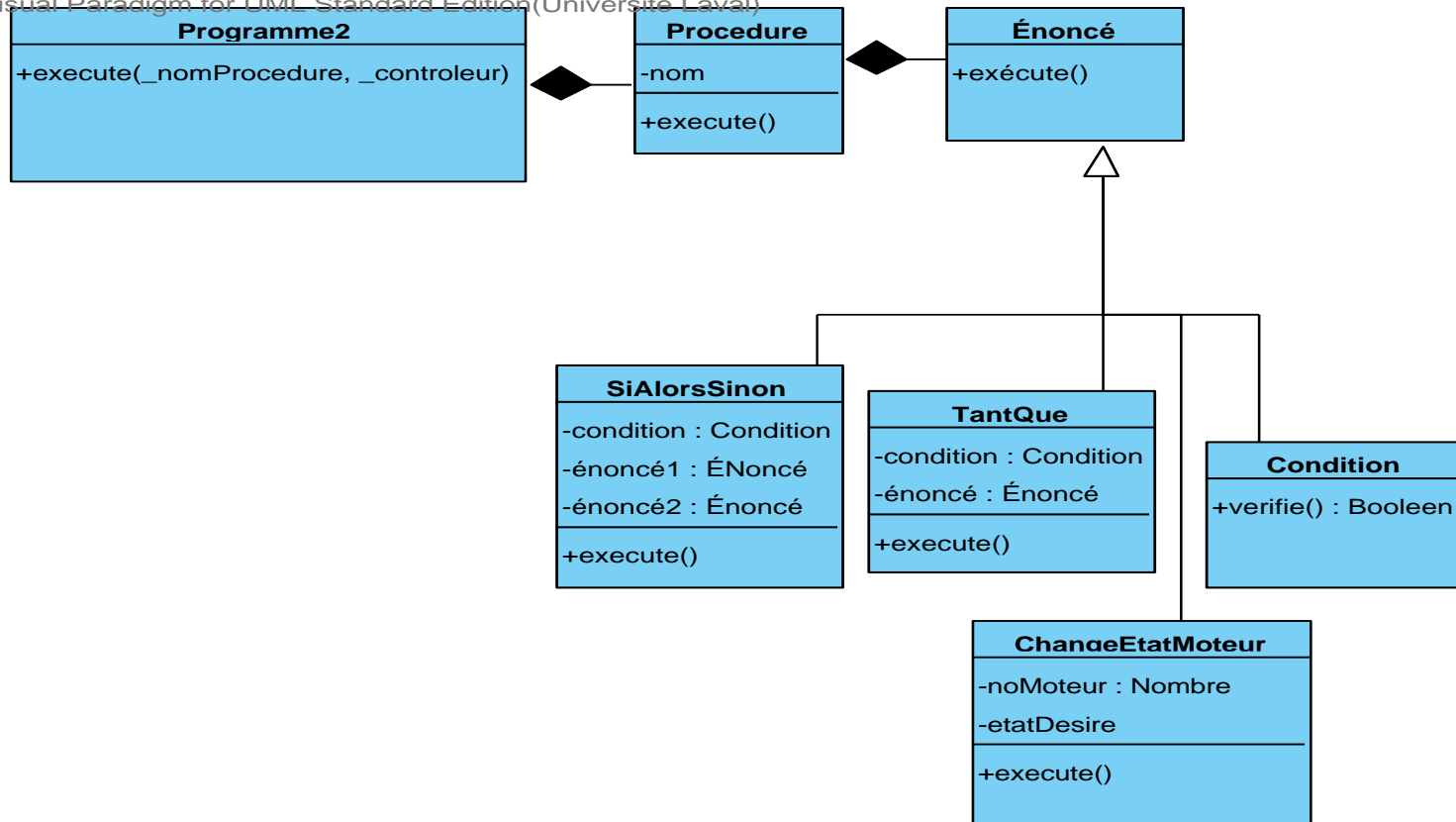
Sinon, moteur 1 négatif

Attends 5 secondes

Moteur 1 arrêté

# Exemple: Robo sapiens

Visual Paradigm for UML Standard Edition (Université Laval)



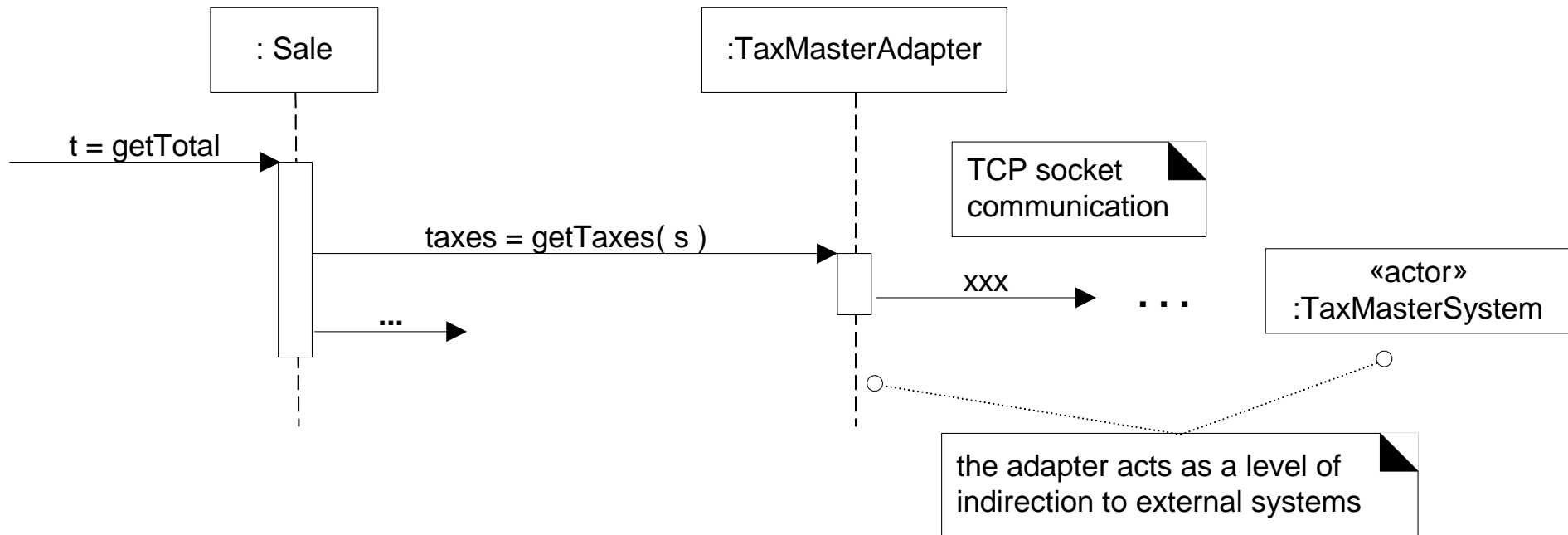
# Principe 8 : Indirection

- **Problème:**

- Où affecter les responsabilités pour éviter le couplage entre deux entités (ou plus) ?
- Comment découpler les objets pour maintenir le potentiel de réutilisation?

- **Solution:**

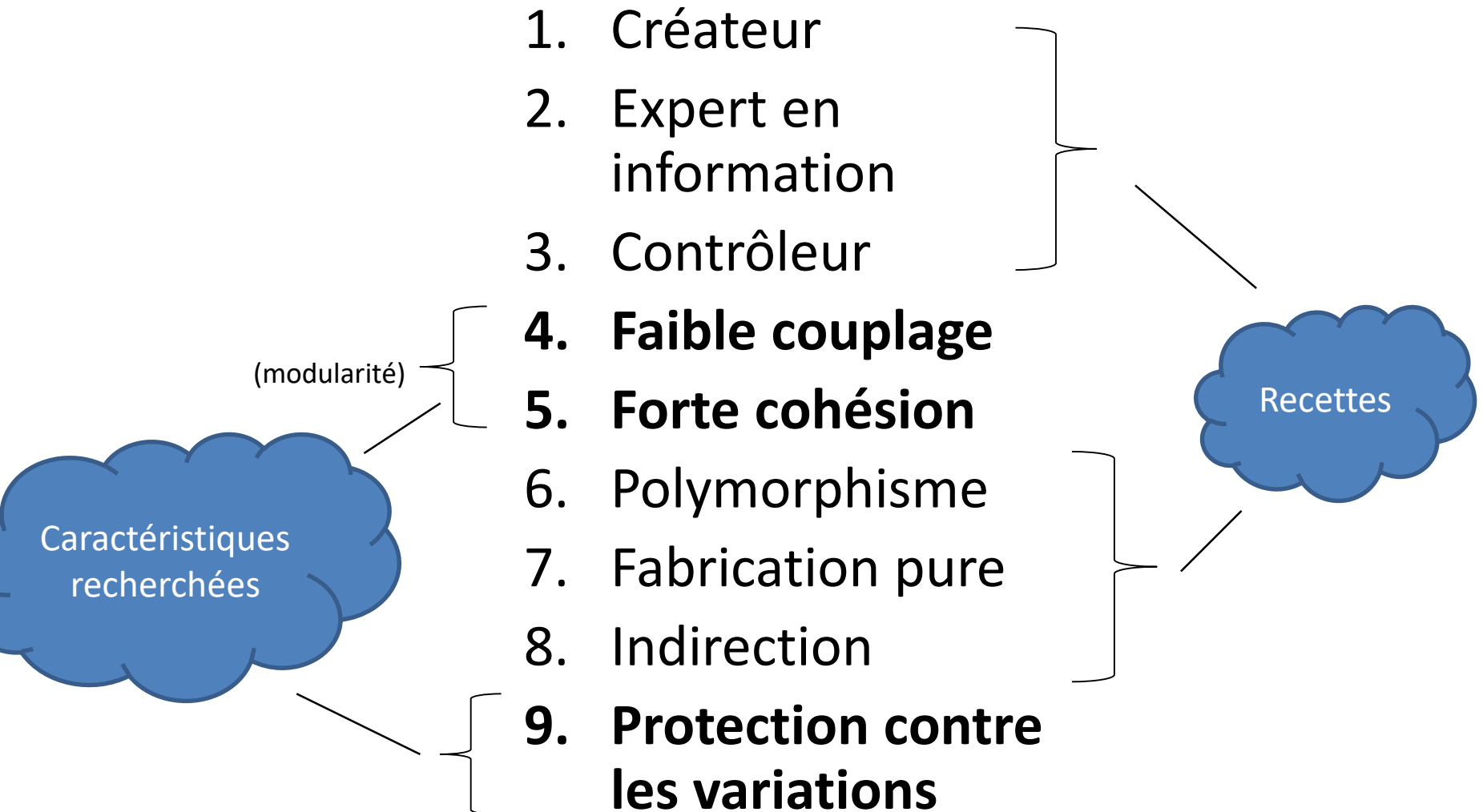
- Affecter la responsabilité à un objet qui sert d'intermédiaire entre d'autres composants ou systèmes externes pour éviter de les coupler directement
- L'intermédiaire crée une indirection entre les composants



# Robo sapiens

- Gestionnaire des accès au matériel (capteurs, moteurs, etc.)
  - Même si le fabricant du matériel vous donne des classes pour manipuler son matériel, ne laissez pas votre code utiliser directement ces classes
  - Insérez une classe intermédiaire entre les deux (de manière à créer une indirection)
  - Si un jour vous changez le matériel, vous n'aurez alors qu'UNE SEULE CLASSE À MODIFIER pour que toute votre application (ou même toutes vos applications) soient compatibles avec ce nouveau matériel.

# Grands principes (GRASP)



# À faire pour ce module

- Lecture des chapitres
  - Version anglaise: 25,26
  - Version française: 22,23
- Comprendre les grands principes de design orienté objet (GRASP)
- Projet de session