Programmation orientée objet SOLID

Bruno MERMET

2010

SOLID

- Introduction
  - Acronyme introduit au début des années 2000 par Robert Cecil Martin
  - Résume 5 principes clés à appliquer en POO pour produire du code facile à maintenir

- Signification
  - S : Single Responsability Principle
  - O : Open/Closed Principle
  - L : Liskov Substitution Principle
  - I : Interface Segragation Principle
  - D : Dependency Inversion Principle

Single Responsability Principle (SRP)
Principe de Responsabilité Unique

- Principe

Chaque objet est en charge d'une seule *responsabilité*, laquelle doit être complètement encapsulée dans la classe.

- Responsabilité = raison de changer (R. C. Martin)

- Corollaire

Une classe ne doit être susceptible de changer que pour une seule raison.

Open/Closed Principle (OCP)
Principe d'ouverture/fermeture

- Principe

Les entités logicielles (classes, modules, fonctions, etc.) doivent être ouvertes aux extensions, mais fermées aux modifications.

Version de B. Meyer

Une classe ne peut être modifiée que pour corriger des erreurs. L'ajout de nouvelles fonctionnalités ne peut se faire que par la création de nouvelles classes (par exemple des sous-classes de la première).

Version polymorphique

On utilise au maximum les interfaces, figées. Mais elles peuvent être implantées librement et augmentées par héritage.

Liskov Substitution Principle (LSP)
Principe de Substituion de Liskov

- Principe (formalisé en 1994 par Liskov & Wing)

Soit q(x) une propriété vérifiable par tous les objets x de type T. Alors q(y) doit être vrai pour tous les objets y de S où Y est un sous-type de T.

- Mise en oeuvre
  - En partie via le langage
  - En partie en s'assurant des propriétés suivantes :
    - Les préconditions ne peuvent pas être renforcées dans un sous-type
    - Les post-conditions ne peuvent pas être affaiblies dans un sous-type
    - L'invariant du super-type doit être préservé dans un sous-type
- Corollaire

Un carré n'est pas forcément un sous-type d'un rectangle (setLargeur ne préserve pas la hauteur par exemple)

Interface Segragation Principle (ISP)
Principe de ségrégation des interfaces

- Principe

Une interface doit ne comporter que des méthodes en rapport avec l'interface elle-même, de façon à ce que les clients d'une interface ne connaissent que les méthodes en rapport avec cette interface.

- Corollaire

Aucune client d'une interface ne doit dépendre de méthodes (de l'interface) qu'il n'utilise pas.

Dependency Inversion Principle (DIP)
Principe d'inversion des dépendances

- Principe

Dans les architectures classiques, les composants de haut niveaux dépendent des composants de bas niveau sur lesquels ils reposent. Le DIP établit au contraire que :

  - A. Les modules de haut niveau ne doivent pas dépendre des modules de bas niveau. Les deux doivent dépendre d'abstractions
  - B. Les abstractions ne doivent pas dépendre de détails, mes les détails doivent dépendre des abstractions

- Mise en oeuvre
  - Les aspects "bas niveau" sont représentés dans le composant de haut niveau par des interfaces
  - Les composants "bas niveau" doivent implanter les interfaces requises (on peut utiliser le pattern Adaptateur pour ce faire)