Révisons ensemble les principes SOLID

Au travers

d'un petit atelier ludique (création originale d'Olivier AZEAU, Version raccourcie)

et d'une revue de code rapide, ou on tentera d'illustrer l'utilisation de ces principes dans notre code-tidien.

Si t'es pas SOLID t'es pas agile !

- by Olivier Azeau
- Des membres de l'assistance viennent sur scène pour jouer le rôle de composants logiciels et ainsi découvrir
  comment améliorer les relations entre composants pour faciliter la maintenance de l'ensemble du logiciel
- Après une introduction, l'animateur explique les règles du jeu.
  La suite est plus ou moins prévisible. L'idée est de passer par des configurations où les composants sont dans une situation précise,
  chacune illustrant un principe SOLID.

REGLES DU JEU

Règle	Intention sous-jacente
Les composants peuvent exécuter tous les ordres qui sont donnés mais ne savent pas parler	Le langage humain permet des interactions trop évoluées qui ne reflètent pas les liaisons entre composants au sein d'un logiciel
Un composant ne peut pas être partiellement reprogrammé : si ses ordres changent, on doit les réexpliquer en entier	Décourager des modifications trop détaillées qui empêchent l'émergence des abstractions au sein du logiciel.
Seule exception : si un collaborateur d'un composant est reprogrammé, il suffit d'indiquer au composant le nouveau collaborateur	Simuler l'association entre 2 objets qui ne change pas si l'implémentation de l'un deux change
On peut programmer un composant en lui disant simplement "fait la même chose que lui«	Simuler l'instanciation d'un objet à partir d'une classe

PRINCIPES
SOLID

- **S**INGLE **R**ESPONSABILITY **P**RINCIPLE (**SRP**)
- **O**PEN/**C**LOSE **P**RINCIPLE (**OCP**)
- **L**ISKOV'S **S**UBSTITUTION **P**RINCIPLE (**LSP**)
- **I**NTERFACE **S**EGREGATION **P**RINCIPLE (**ISP**)
- **D**EPENDENCY **I**NVERSION **P**RINCIPLE (**DIP**)

SINGLE RESPONSABILITY PRINCIPLE (SRP)

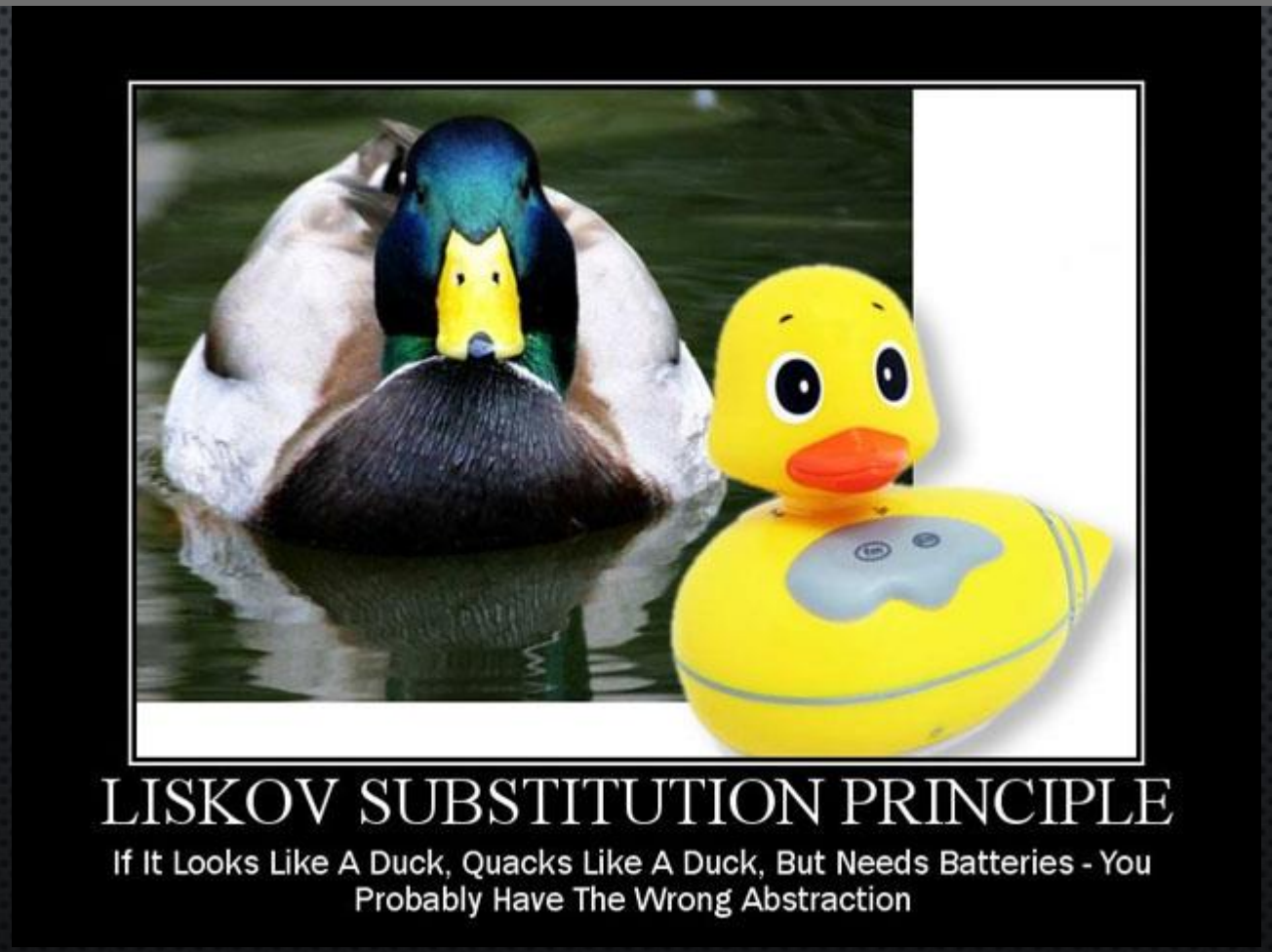
A CLASS SHOULD HAVE ONLY ONE REASON TO CHANGE.

OPEN/CLOSE PRINCIPLE (OCP)

SOFTWARE ENTITIES (CLASSES, MODULES, FUNCTIONS, ETC.)
- SHOULD BE OPEN FOR EXTENSION
- BUT CLOSED FOR MODIFICATION.

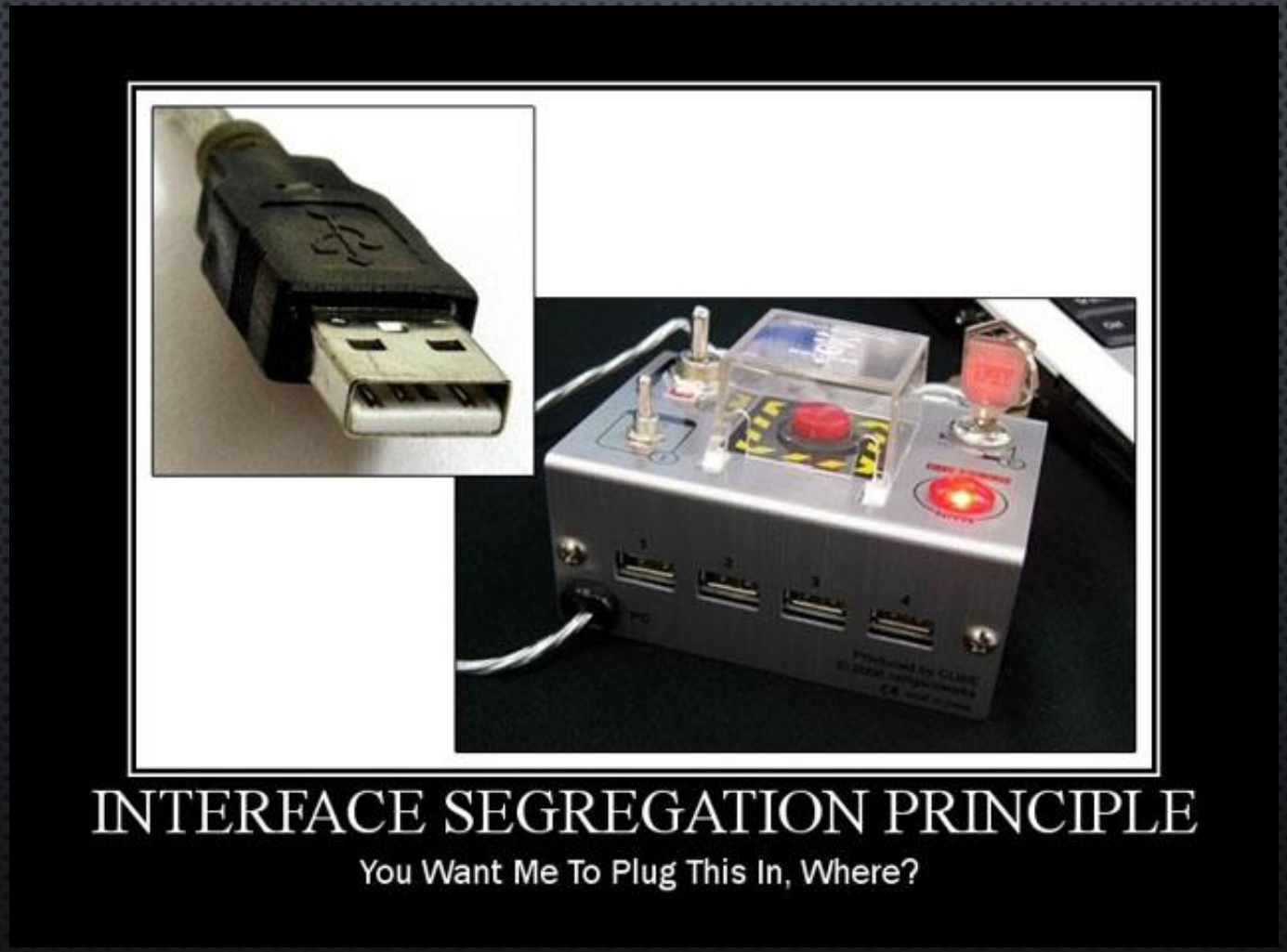
LISKOV'S SUBSTITUTION PRINCIPLE (LSP)

FUNCTIONS THAT USE POINTERS OR REFERENCES TO BASE CLASSES MUST BE ABLE TO USE OBJECTS OF DERIVED CLASSES WITHOUT KNOWING IT.



INTERFACE SEGREGATION PRINCIPLE (ISP)

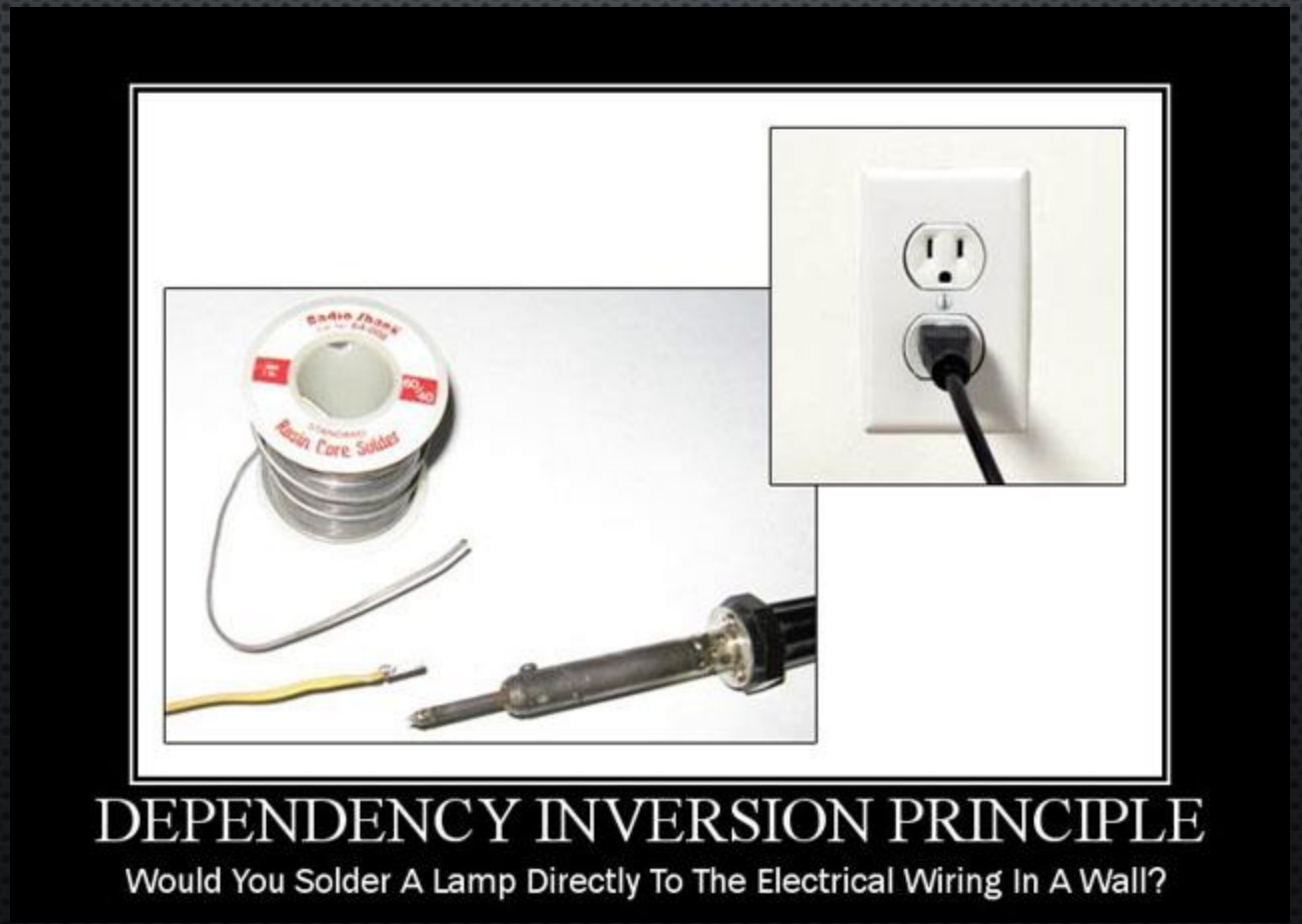
CLIENTS SHOULD NOT BE FORCED TO DEPEND UPON INTERFACES THAT THEY DO NOT USE.



DEPENDENCY INVERSION PRINCIPLE (DIP)

A. HIGH LEVEL MODULES SHOULD NOT DEPEND UPON LOW LEVEL MODULES. BOTH SHOULD DEPEND UPON ABSTRACTIONS.

B. ABSTRACTIONS SHOULD NOT DEPEND UPON DETAILS. DETAILS SHOULD DEPEND UPON ABSTRACTIONS.



CODES EXEMPLES

- CF : https://github.com/Squallygator/solid/