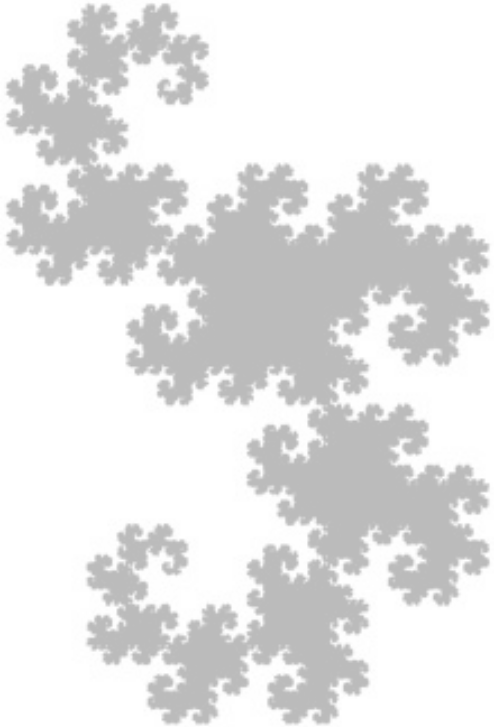
Шаблони проектування

від GRASP до SOLID

Що таке Шаблон Проектування?

Патерн - іменований опис типової задачі та розв’язку, що може бути застосований до нового контексту;

містить поради щодо застосування в різних обставинах, включаючи переваги й недоліки



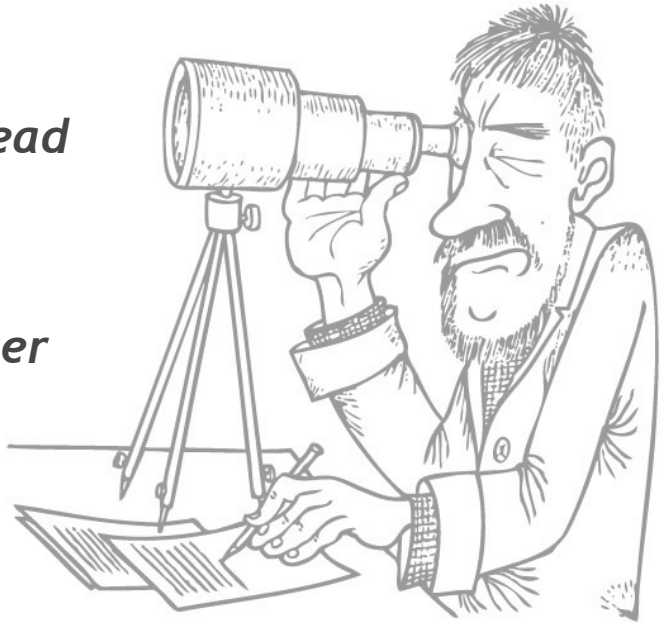
- Повторюваність
- ~~Новий патерн~~
- Комунікативна роль

Трохи історії

- 1977 - Christopher Alexander “A Pattern Language”
- 1987 - Kent Beck & Ward Cunningham застосували ідеї шаблонів для написання UI на Smalltalk
- 1994 - Gamma, Helm, Johnson & Vlissides (GoF) “Design Patterns”
- 1997 - Craig Larman “Applying UML and Patterns” GRASP
- 2000 - Robert Martin сформулював SOLID

Різновиди Патернів

- *Gang of Four: породжуючі, структурні, поведінкові; всього 23*
- *GRASP - патерни розподілу обов'язків*
- *Багатопоточного програмування: Lock, Thread Pool, Monitor*
- *Проектування архітектури: MVC, N-шарова*
- *Інтеграційні патерни: Message Router, Splitter*
- *Java EE: DAO, Session Facade, Service Locator*
- *Антипатерни: God object, Spaghetti code*
- *інші…*



General Responsibility Assignment Software Patterns

GRASP - опис фундаментальних принципів об'єктного проектування та присвоєння обов'язків, у вигляді патернів

Обов'зки

Робити

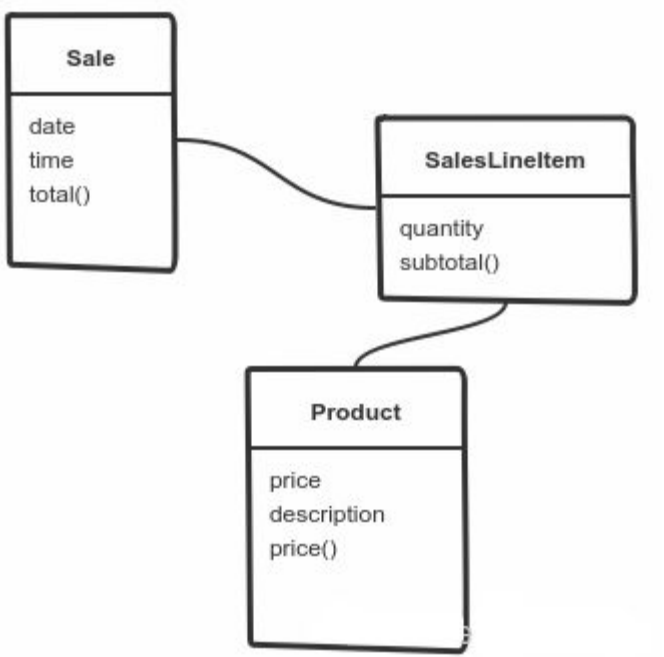
- щось власне, розрахунки
- ініціювати дію в інших об'єктах
- керувати/координувати активності в інших об'єктах

Знати

- внутрішні, інкапсульовані дані
- стосовно пов'заних об'єктів
- щодо наслідків або результатів розрахунків

Register -> Point-of-Sale Terminal

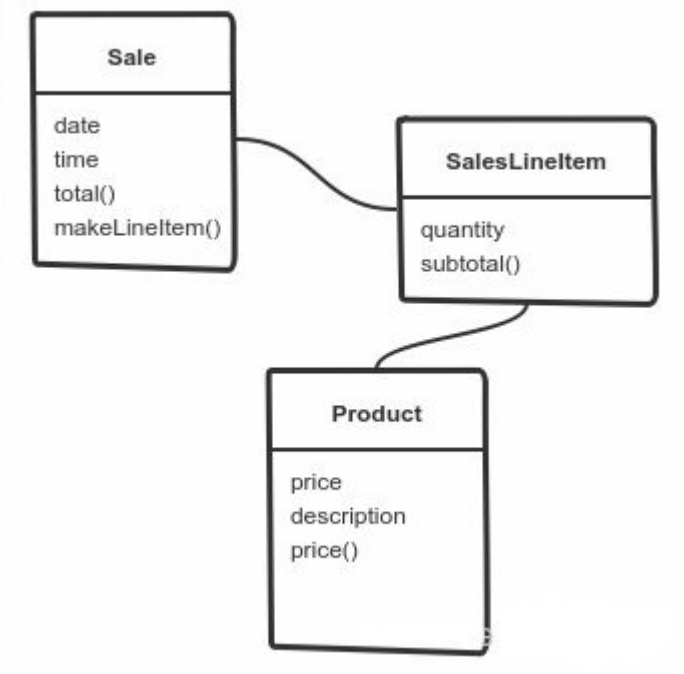
Information Expert



Задача: який найбільш загальний принцип розподілу обов'язків

Розв'язок: призначити обов'язки інформаційному експерту, тобто класу, що містить необхідну інформацію

Creator

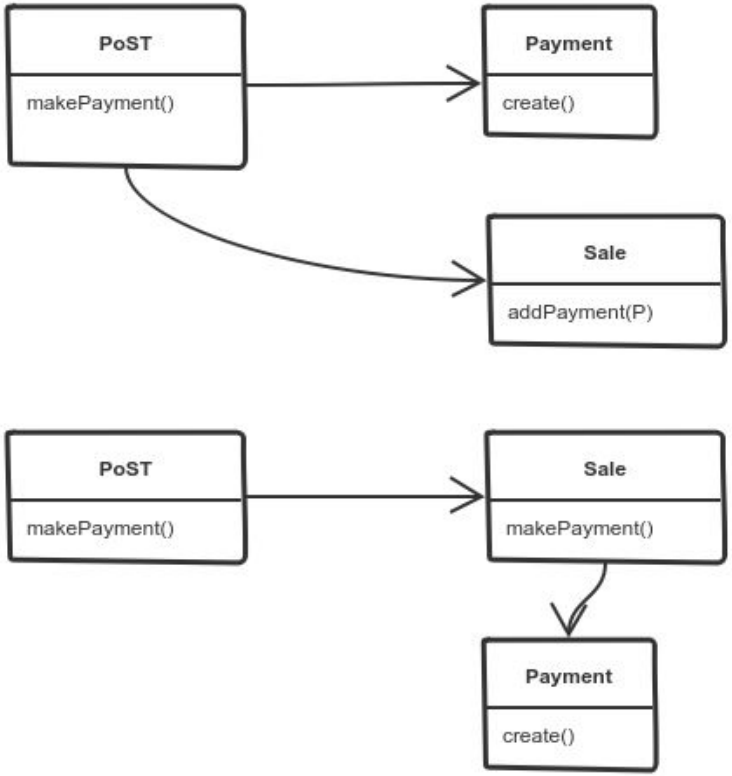


Задача - хто повинен нести відповідальність за створення нового об'єкту

Розв'язок - клас B створює екземпляри класу A у випадку

- *B аґреґує об'єкти A*
- *B містить об'єкти A*
- *B записує екземпляри об'єктів A*
- *B активно використовує A*
- *B має дані ініціалізації A*

Low Coupling - Слабка зв’язаність

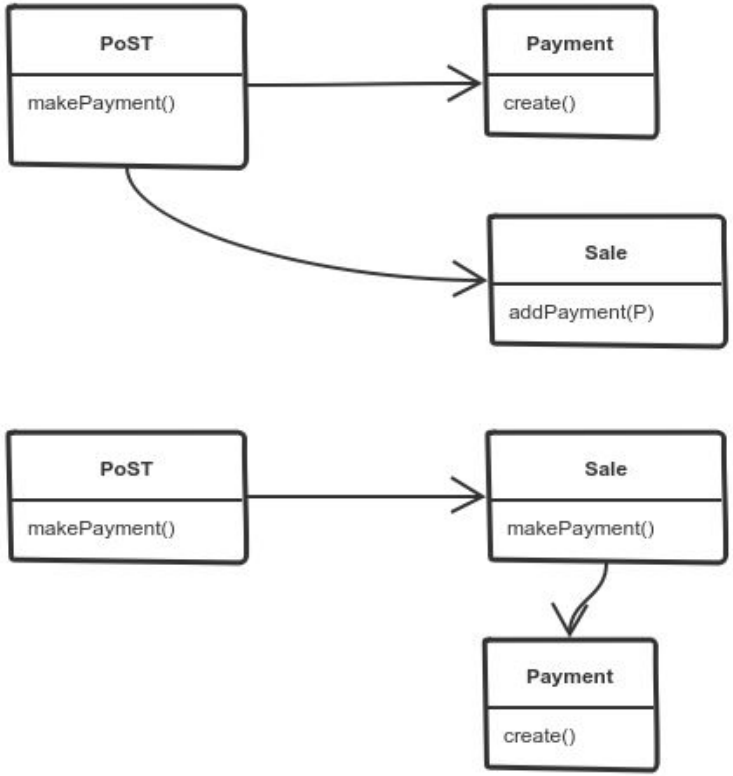


Проблема *- клас, що сильно зв’язаний з багатьма іншими*

- *зміни у зв’язаних класах призводять до локальних змін в поточному*
- *ускладнюється розуміння коду*
- *унеможливлюється повторне використання*

Розв’язок *- розподілити обов’язки таким чином, щоб степінь зв’язаності залишалась низькою*

High Cohesion - Сильне функціональне зчеплення



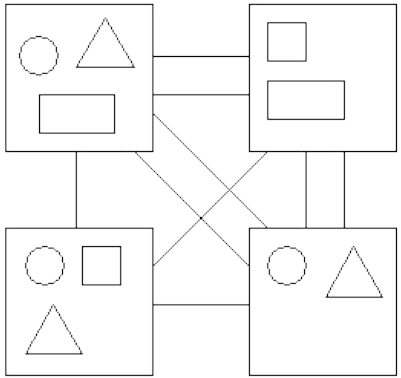
Проблема *- як стримувати зростаючу складність системи?*

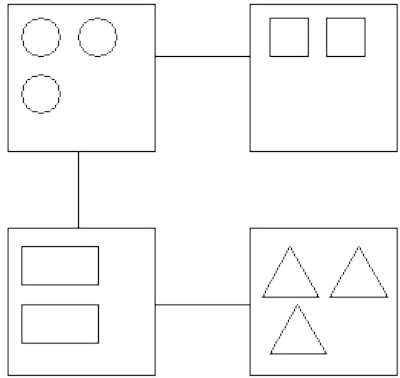
Розв'язок *- розподіляти обов'язки таким чином, щоб функціональне зчеплення залишалася високим*

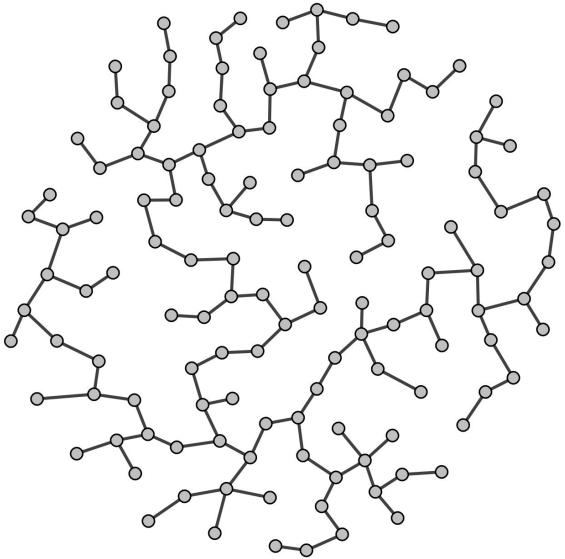
Cohesion (зчеплення) *- міра зв'язності та зфокусованості обов'язків класу*

Coupling vs Cohesion

- *Ін та Ян проектування*
- *Виправданий випадок слабкого зчеплення: оптимізація кількості дорогих запитів задля продуктивності системи*







Модульність *- властивість системи, що була декомпозована на зчеплені і слабо зв'язані модулі*

Controller

Проблема - хто має обробляти зовнішні події?

Controller -

- *об'єкт, що не належить UI області і*
- *відповідальний за обробку зовнішних подій*
- *делегує та координує активності, не містить бізнес-логіки*

Розв'язок - відповідальним за обробку може бути:

- *Facade Controller - представляє систему загалом*
- *<UseCase>Handler, <UseCase>Session - представляє окремі сценарії*

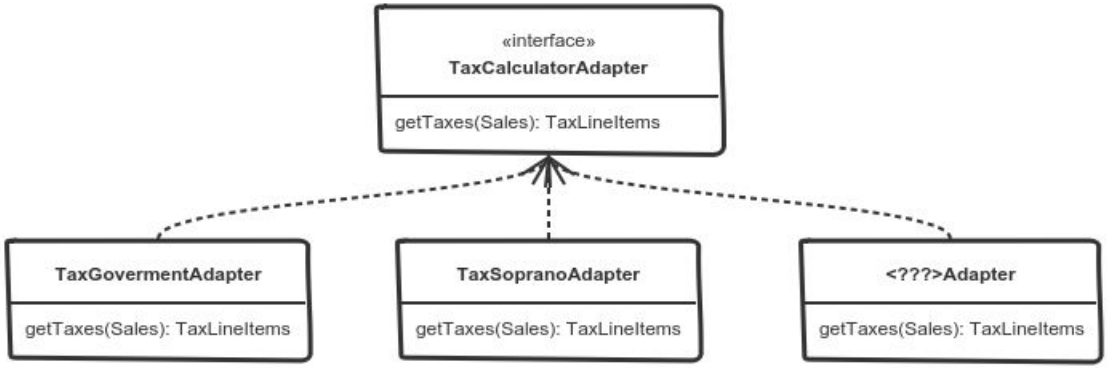
Polymorphism

Задача:

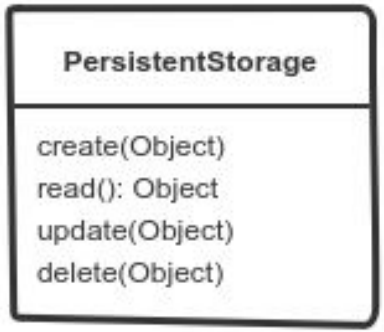
- *як працювати з альтернативними, що різняться типом?*
- *як написати підключаємі компоненти?*

Розв'язок:

- *замінять if-then-else поліморфними методами*



Pure fabrication - Чиста вигадка



Проблема *- як зберегти coupling&cohesion за відсутності підходящих сутностей в домені?*

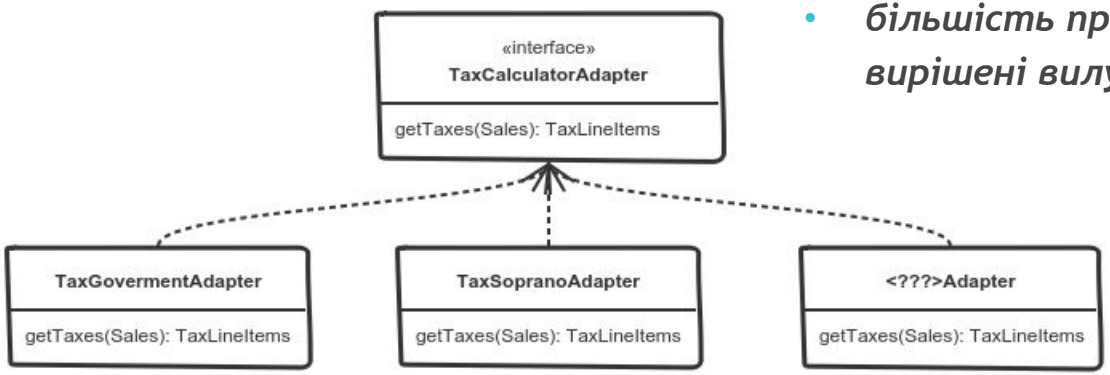
Розв'язок *- вигадати штучний об'єкт, що буде зберігати гарні показники coupling&cohesion*

Indirection - Опосередкований

<u>Задача</u> - розв'язати об'єкти з можливістю повторного використання

<u>Розв'язок</u> - наділяти обов'язками проміжні, опосередковані об'єкти

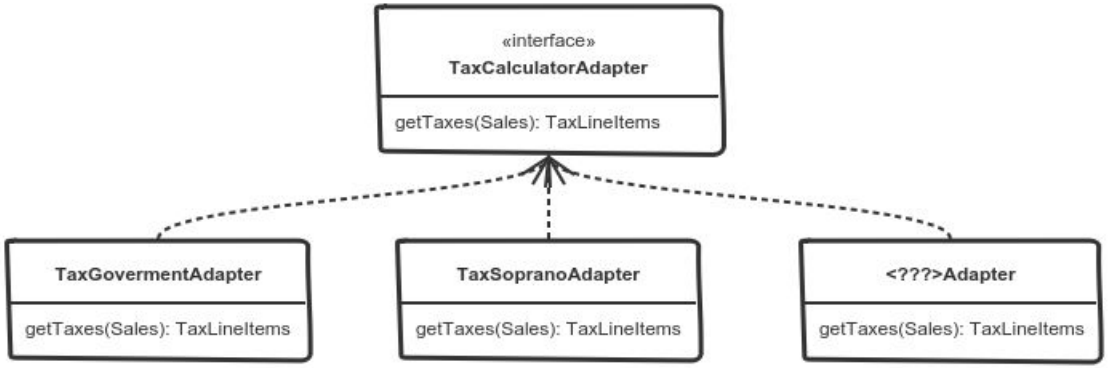
- *більшість проблем Computer Science можуть бути вирішені шляхом введення додаткового рівня абстракції*
- *більшість проблем з перформансом можуть бути вирішені вилученням зайвого рівня абстракції*



Protected variations - Стійкий до змін

Проблема *- як розробити систему, щоб зміни частини системи не афектили решту?*

Розв'язок *- ідентифікувати можливі місця майбутніх змін і надати їм обов'язки для побудови сталого інтерфейсу навколо них*

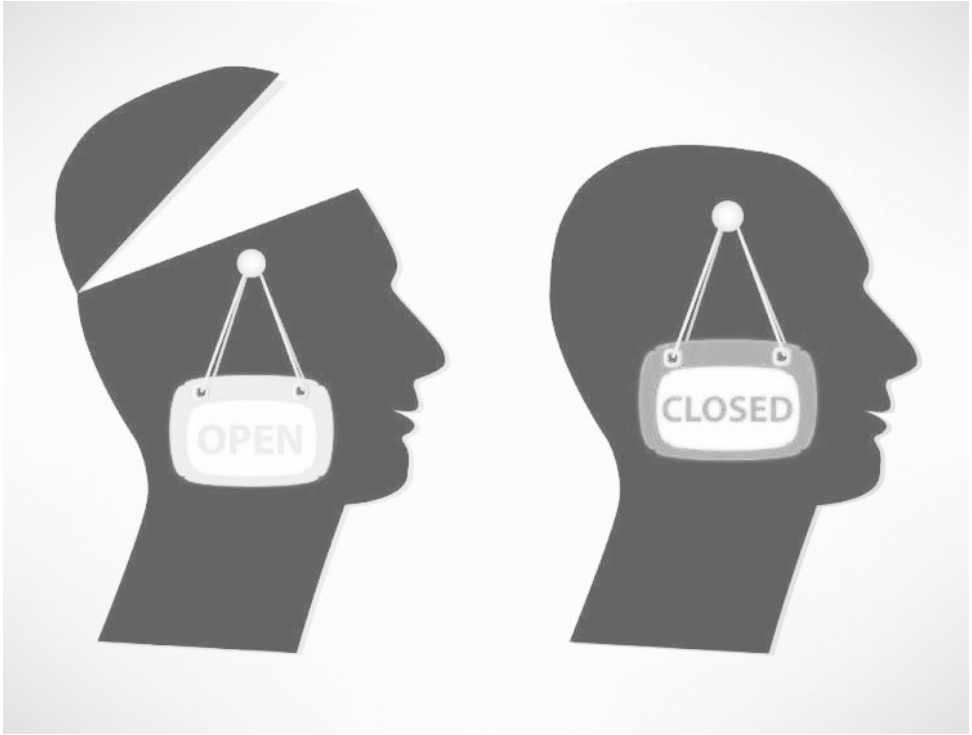


Single responsibility principle - Єдиного обов’язку

SRP: Кожен об'єкт має виконувати лише один обов'язок.

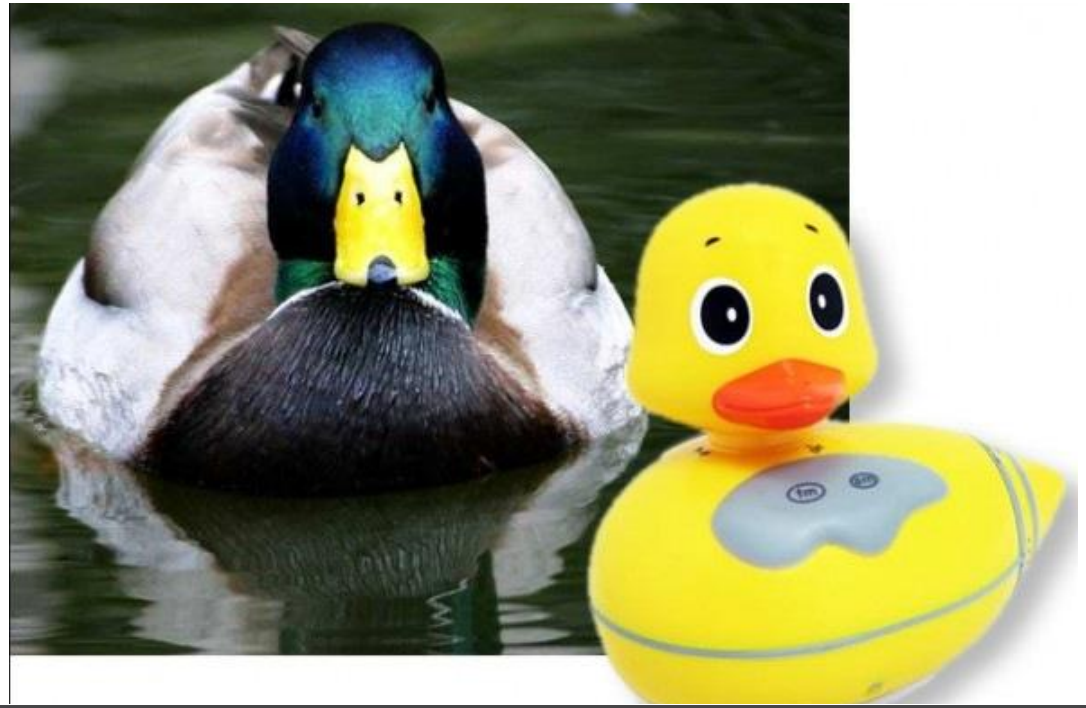
Open/closed principle - Відкритості/закритості

OCP: Програмні сутності повинні бути відкритими для розширення, але закритими для змін.



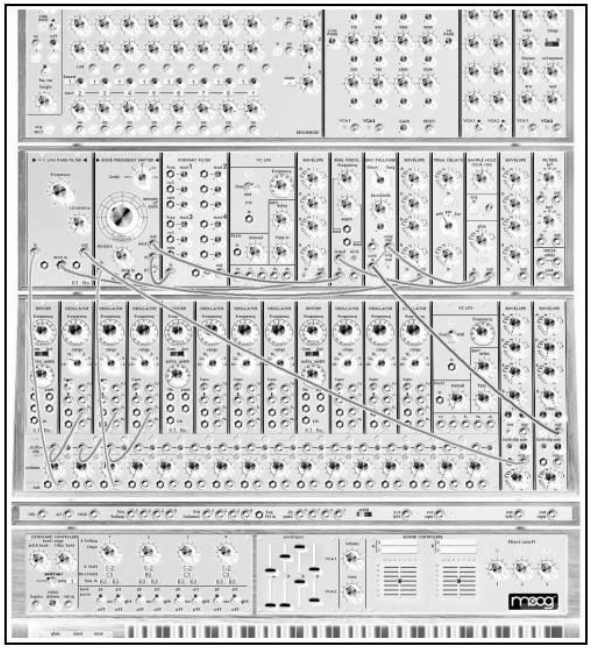
Liskov substitution principle - підстановки Барбари Лісков

LSP: Об'єкти в програмі можуть бути замінені їх нащадками без зміни коду програми.



Interface segregation principle - розділення інтерфейсу

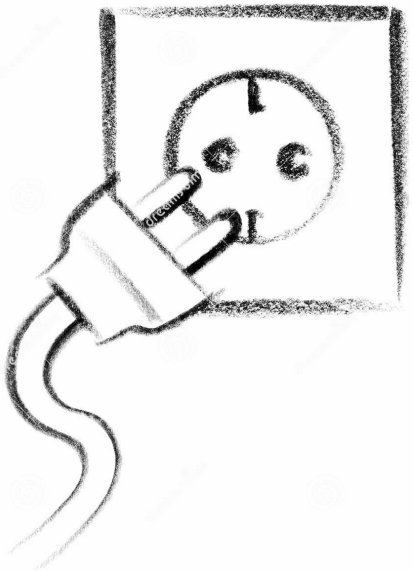
ISP: Багато спеціалізованих інтерфейсів краще за один універсальний.



Dependency inversion principle - інверсії залежностей

DIP: Залежності всередині системи будуються на основі абстракцій, що не повинні залежати від деталей; навпаки, деталі мають залежати від абстракцій.

Модулі вищих рівнів не залежать від модулів нижчих рівнів.



Questions?