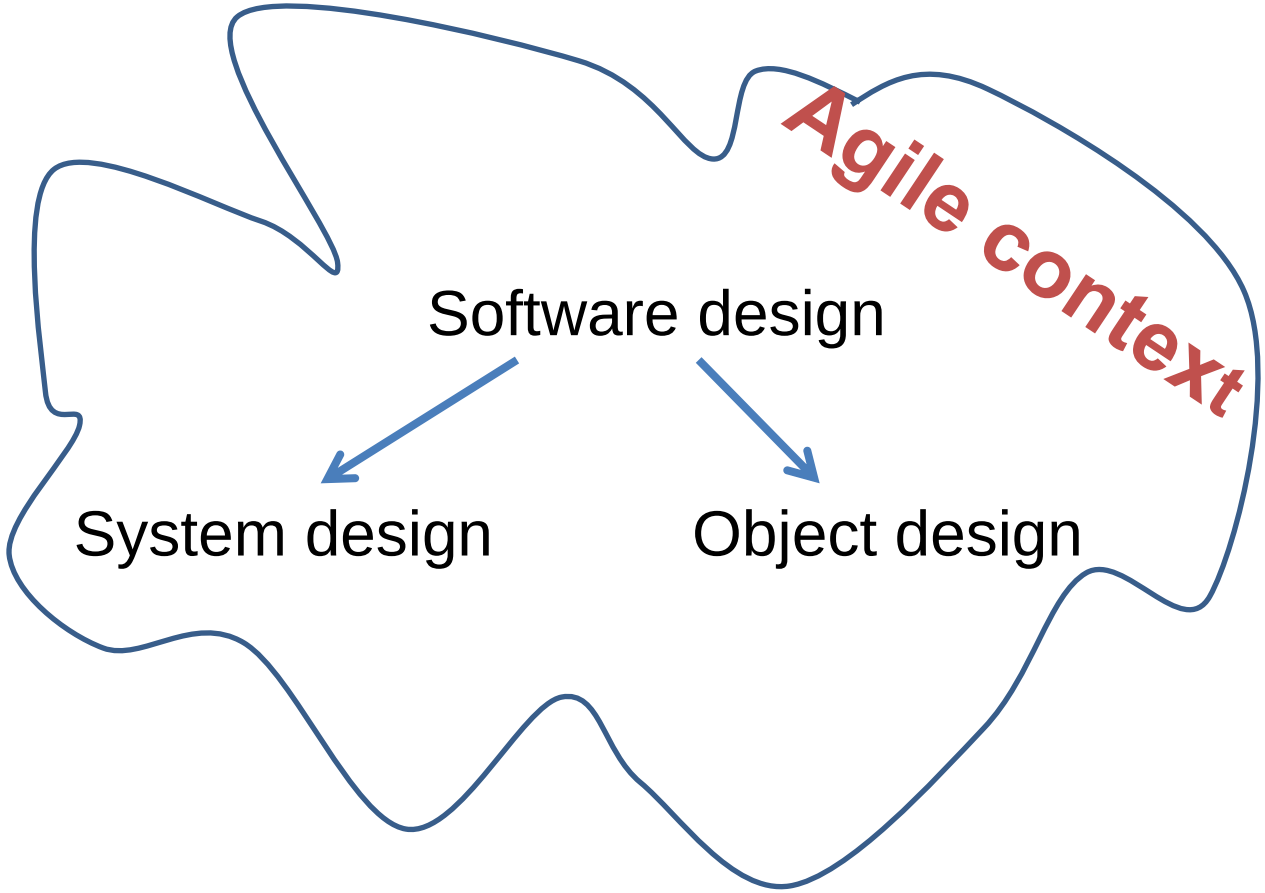
Software Design

Datamatiker /Computer Science

2nd Semester

Spring 2017

Software Design



Design Principles

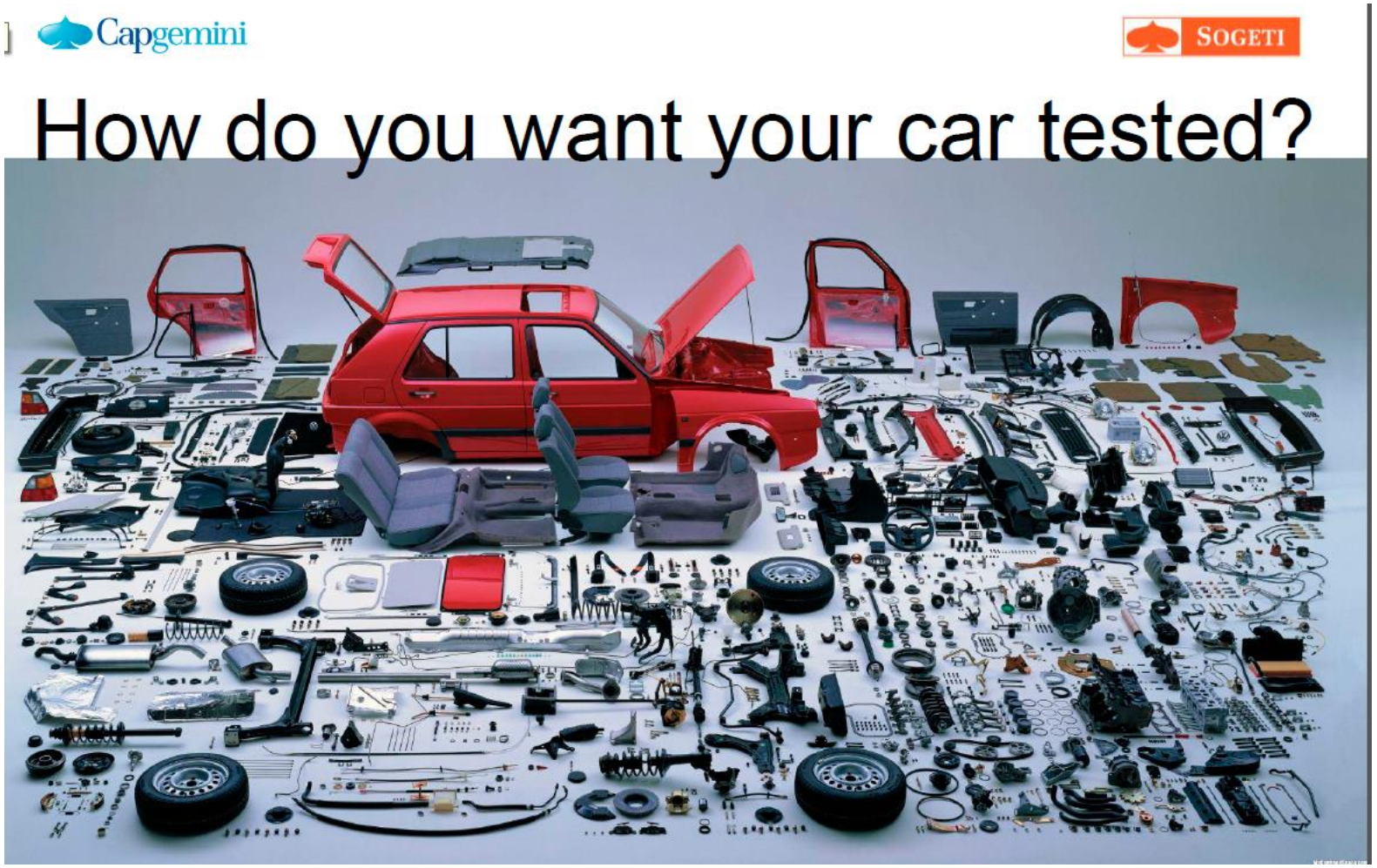
1. Divide and conquer
2. High cohesion
3. Low coupling
4. High abstraction

Design Principles: Divide and Conquer

- Architecture (decompose system into layers)
- Classes & methods

- Why?
  - Overview & Maintainability
    - We need not deal with (all) high complexity at the same time
  - Reusability
    - e.g. replace UI layer, reuse date function
  - Robustness
    - Each module can be tested individually (layers and classes/methods)

Divide and conquer example



Design Principles: High Cohesion

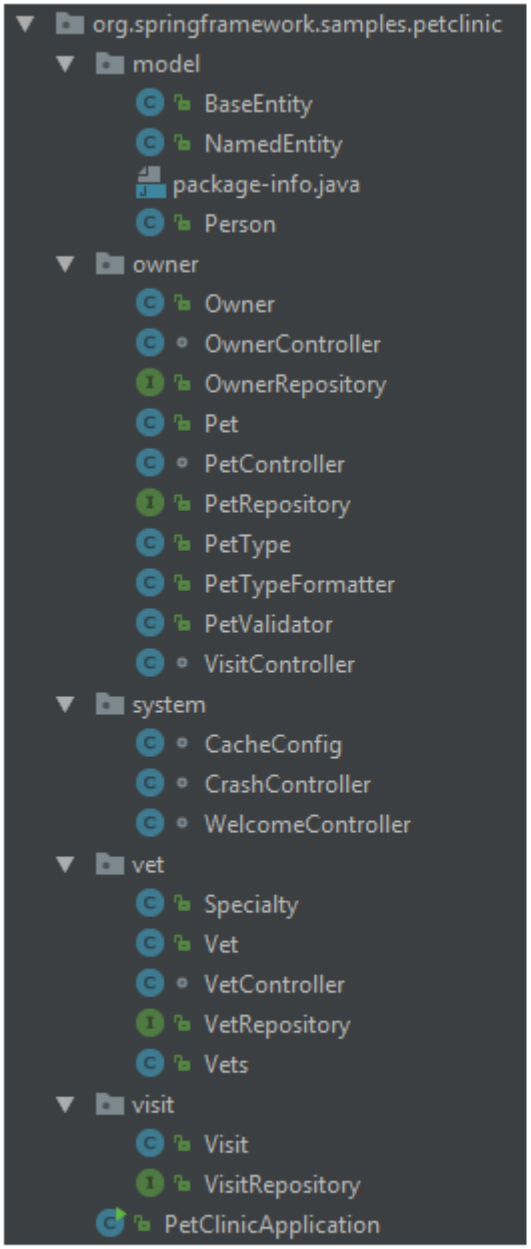
- Architecture (inside layers)
- Class (attributes and methods)

- Why?
  - Reuse and testability (maintainability)
    - Easier to understand and change a module if it keeps together things that are related to each other, and keeps out other things

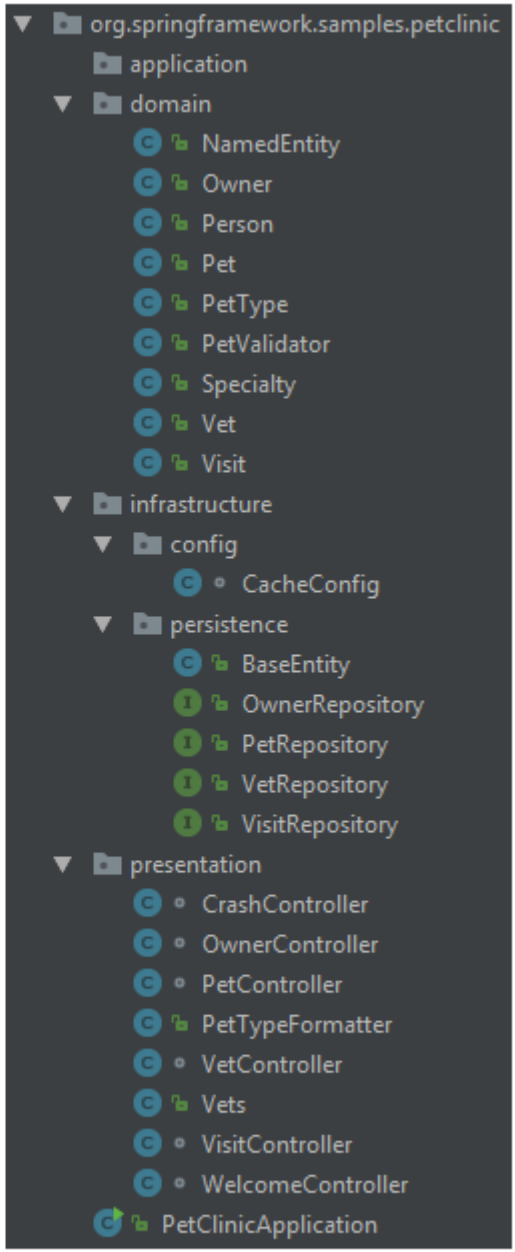
Layers example

2 different ways of layering the same app

The cohesion in the layers is seen from different perspectives in the 2 examples



Spring Pet Clinic Original Architecture



Spring Pet Clinic with Layered Architecture

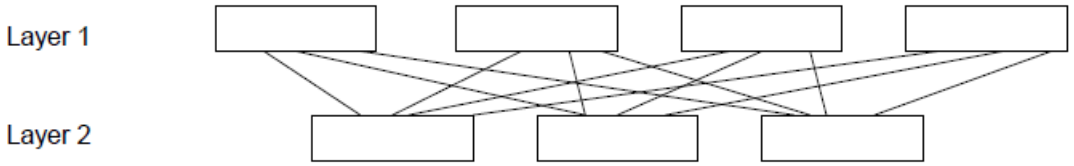
Design Principles: Low Coupling

- Architecture (between layers)
- Between classes inside a layer

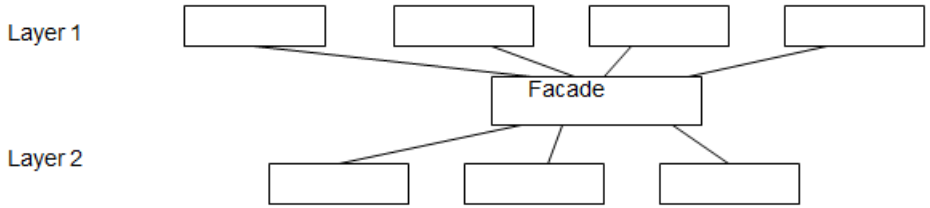
- Why?
  - Reuse and testability (maintainability)
    - When interdependencies exist, changes in one place will require changes somewhere else.
    - A network of interdependencies makes it hard to see at a glance how some component works

Low Coupling examples

High coupling:



Low coupling:



Facade pattern becomes the API of a layer

The Facade class itself ends up with low cohesion

Demo

- Let’s get low coupling between layers ☺

Design Principles: Abstraction

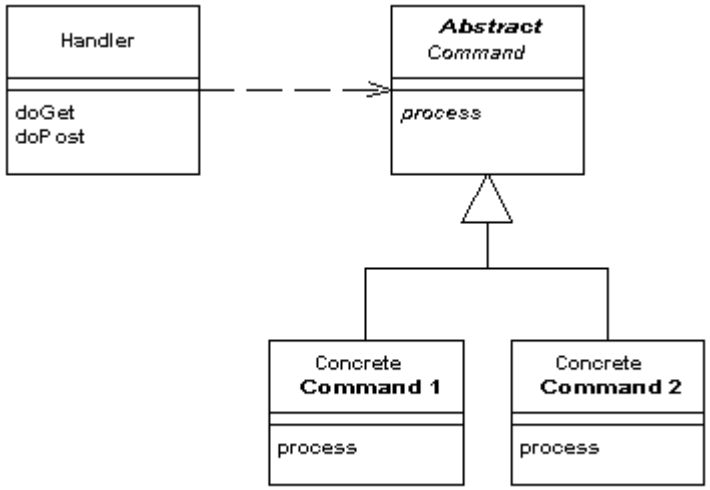
- Architecture (layers)
- Classes & methods

- Why?
  - Provides information hiding
  - Let you understand purpose of module without having to know details

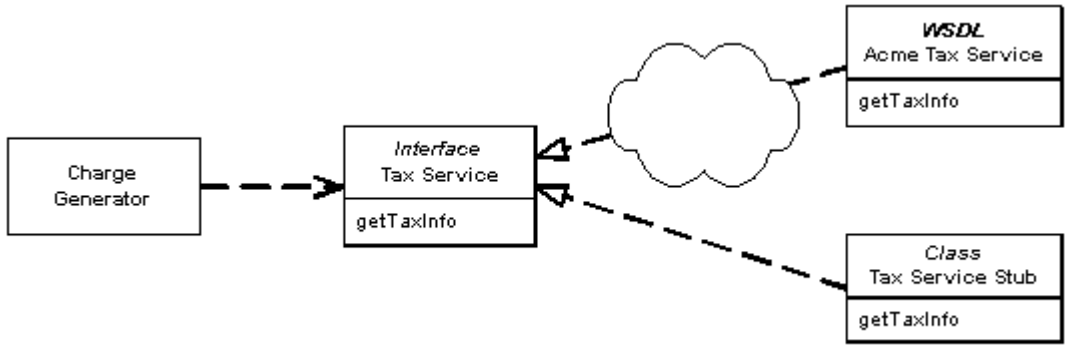
Abstraction examples

- Super classes and interfaces increase level of abstraction

Abstract class ex:



Interface ex:



Exercises

- Put Facade in the data access layer
- Consider Facade and Controller(s) in model layer

- Make sure you have MVC architecture in UI layer
- Consider the Command pattern in the UI layer

Advanced

- Run Source Code metrics for your project
- Consider what interfaces and dependency injection can do for the quality of your project