Object Oriented Analysis & Design
面向对象分析与设计

Lecture_08 通用的职责分配软件原则 GRASP（二）

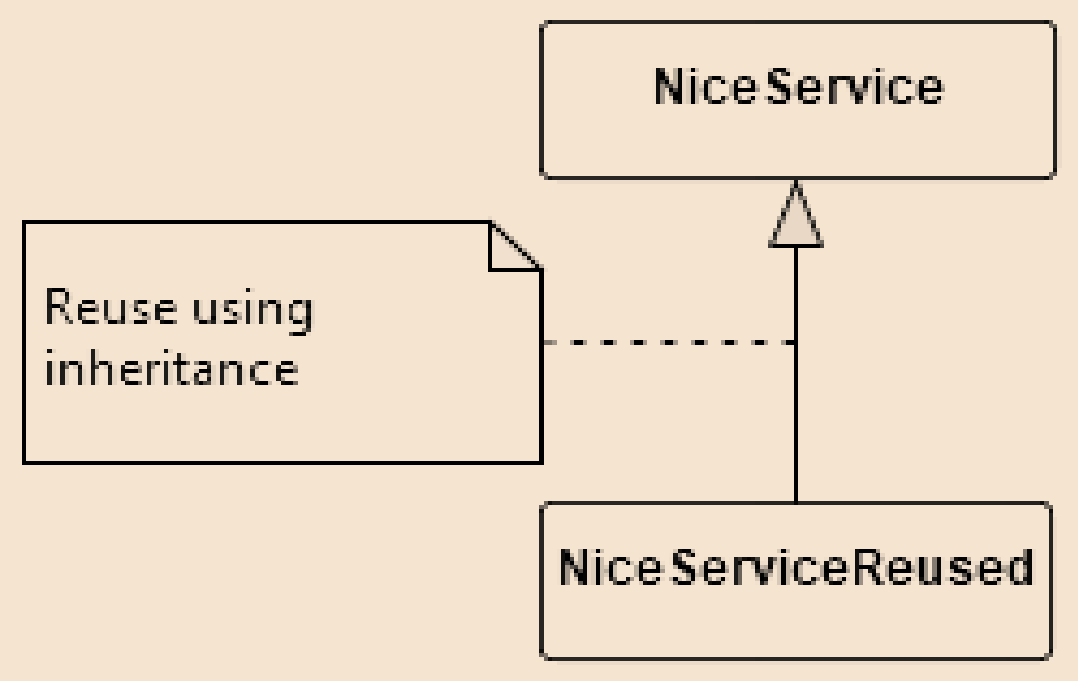
主讲: 姜宁康 博士

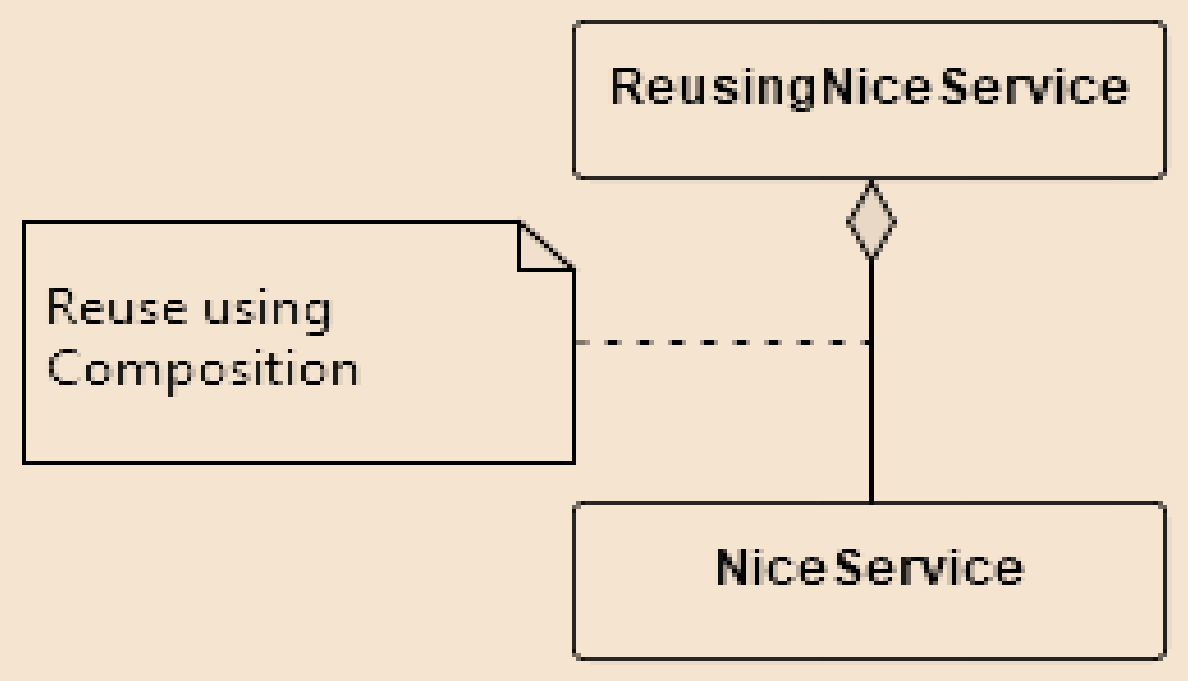
6、其他面向对象设计原则2

- 能用组合的地方，不要用继承 Favor object composition over class inheritance

6.1 代码重用的两种方式

- 能用组合的地方不要用继承 Favor object composition over class inheritance
- 这句话是说，在OO技术里，有两种基本类型的代码重用: 组合、继承 What this statement says is that there are basically two ways to reuse code in OO





6.2 继承的特点

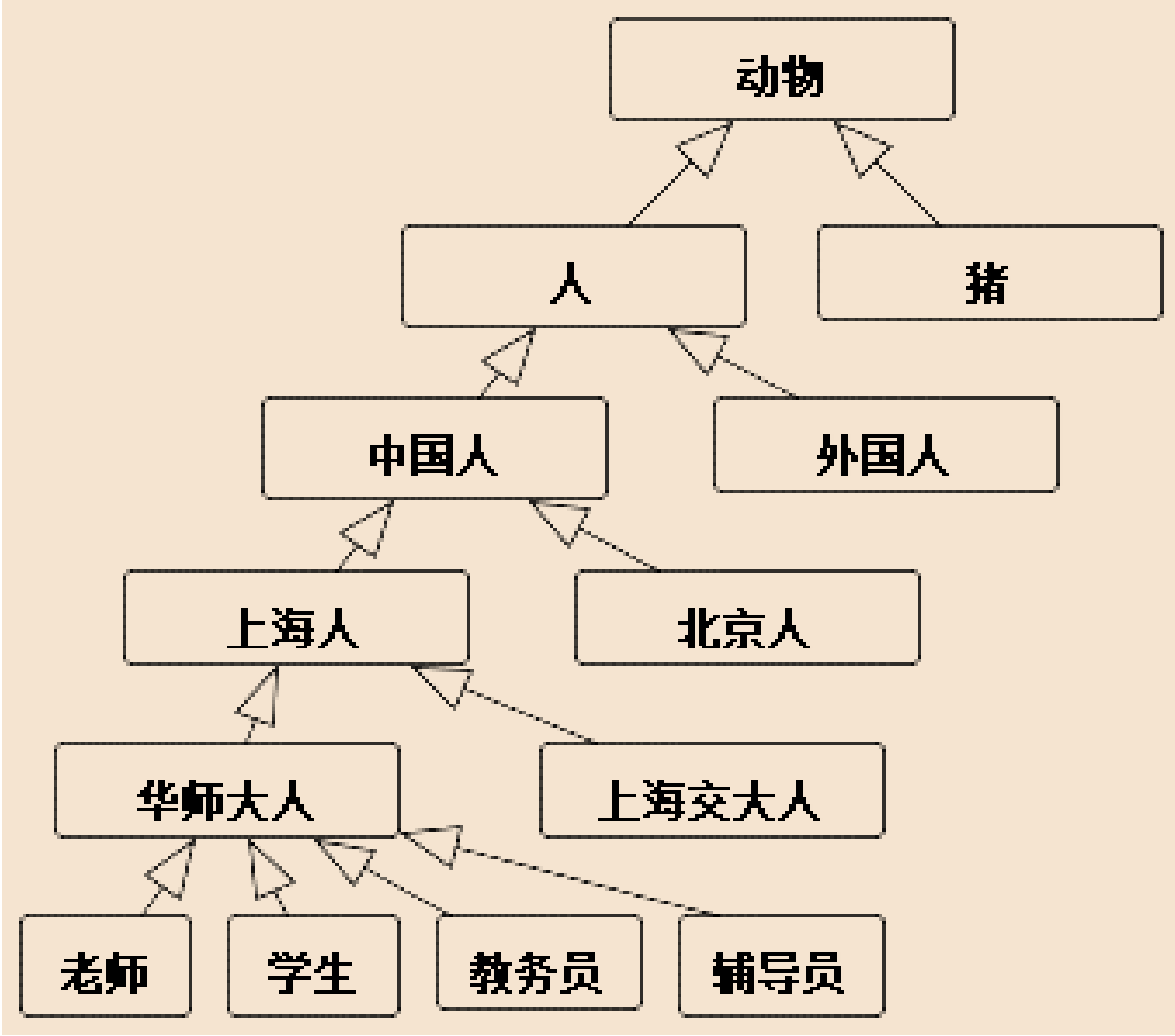
- 类的继承 Class inheritance
  - 子类获得父系“全部功能”，“稍微”调整一下，比如覆盖实现几个方法 You get the “whole packet” and “tweak a bit” by overriding a single or few methods
    - 既快又容易 Fast and easy
    - 代码实现时明确展示，由编程语言加以支持 Explicit in the code, supported by language
      - (you can directly write “extends”)
- 继承存在的副作用...
  - “继承打破了封装性 inheritance breaks encapsulation”
    - 导致父类与子类之间高度耦合 leading to high coupling between super-and subclass ☹
  - 1）继承的代码是静态/编译-时绑定的 Inheritance is a static/compile-time binding
    - 今后改变行为的唯一方法是 edit-compile-debug-edit-compile-debug

6.2 继承的特点

- 继承存在的副作用...
  - 2) 客户需要购买整个软件包 You have to buy the full package !
    - you cannot access only part of the behaviour of the superclass – you get it all
    - For example
      - A nice C++ String implementation in a Library was compiled to have dependencies with 2MB other parts of the library
  - 3)父类定义了许多硬性的规定 Superclasses define (part) of physical representation 例如
    - 父类用了列表数据结构，而子类最好是用树结构 superclass uses a List data structure, but subclass would be much more efficient using a tree structure
    - 子类经常直接使用Public和Protected数据成员，一旦父类修改了这些数据，将会影响全体子类 subclasses usually simple refer to public and protected data members, which means, changing implementation details in superclass effect all subclasses !!!
      - yes – we know :
      - review all subclasses
      - retest all subclasses
    - 不得不了解父类的代码以求真正理解（相比组合，只要了解接口）

6.2 继承的特点

- 关于类的继承
  - 可以使用继承，但设计师必须确信使用继承是很好的解决方案时才使用 Inheritance is OK but you must use it for what it handles really nice
  - 以泛化层次的方式，对概念进行建模 Modelling of concepts in a generalisation hierarchy
    - 不是角色
    - 不是行为
    - 不能随意



6.3 组合的特点

- 对象组合 Object composition
  - 没有打破封装性 No breaking encapsulation !!!
- A) 对象组合是一种动态/运行时绑定 object composition is a dynamic/run-time binding
  - 在运行时切换对象引用，就可以改变行为 you may change behaviour at run-time by "switching" an object reference
- B) 整体与部分之间只有接口边界关联，耦合较低 Loose coupling by respecting the interface borders
  - 对私有数据不存在"窥视与拨弄" No peeking and poking in semi-private data members!
  - 不存在大量无用代码一类的负担 No heavy luggage in the form of data structures that are useless

6.3 组合的特点

- C) 各部分的职责明确 Clear division of responsibilities !!!
  - 每个对象清晰地集中在少量的任务上 each object is clearly focused on a single/few tasks
    - 容易理解 easier to understand、容易维护 easier to maintain
  - 只要阅读接口，就可以了解系统（至少理论上是如此）understanding by reading interfaces only
  - 容易独立测试 easier to test in isolation
    - 使用得当，将导致更可靠的设计
  - 每个类依然“苗条” each class remains small
    - 避免的“巨无霸”这种反模式的现象，即一个类似乎完成了几乎全部的功能... avoiding the “Blob” anti-pattern where one class seems to “suck in” all functionality...
  - 重用的概率大增！larger potential for reuse in other contexts

- 本讲结束