

Ch. 9 设计模式

2018 / 12

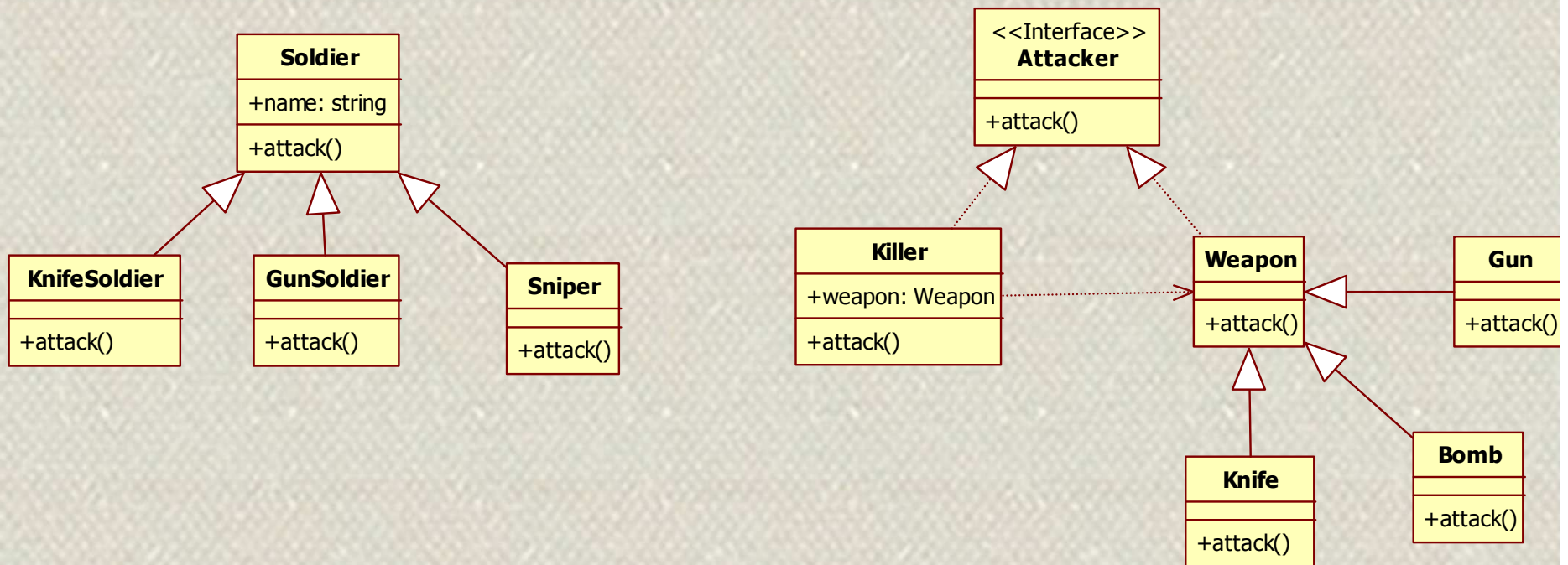
什么是设计模式

- 设计模式(Design Pattern)是一套被反复使用、多数人知晓的、经过分类编目的、代码设计经验的总结，使用设计模式是为了可重用代码、让代码更容易被他人理解、保证代码可靠性。

为什么要使用设计模式

- 实现解耦
- 易于理解
- 易于维护

对比



要点

- 优先使用组合，而不是继承

创建型

- 单例（单件/Singleton）
- 工厂方法（Factory Method）
- 抽象工厂（Abstract Factory）
- 建造者（Builder）
- 原型（Prototype）

结构型

- 适配器(Adapter)
- 桥接(Bridge)
- 组合(Composite)
- 装饰(Decorator)
- 外观(Facade)
- 享元(Flyweight)
- 代理(Proxy)

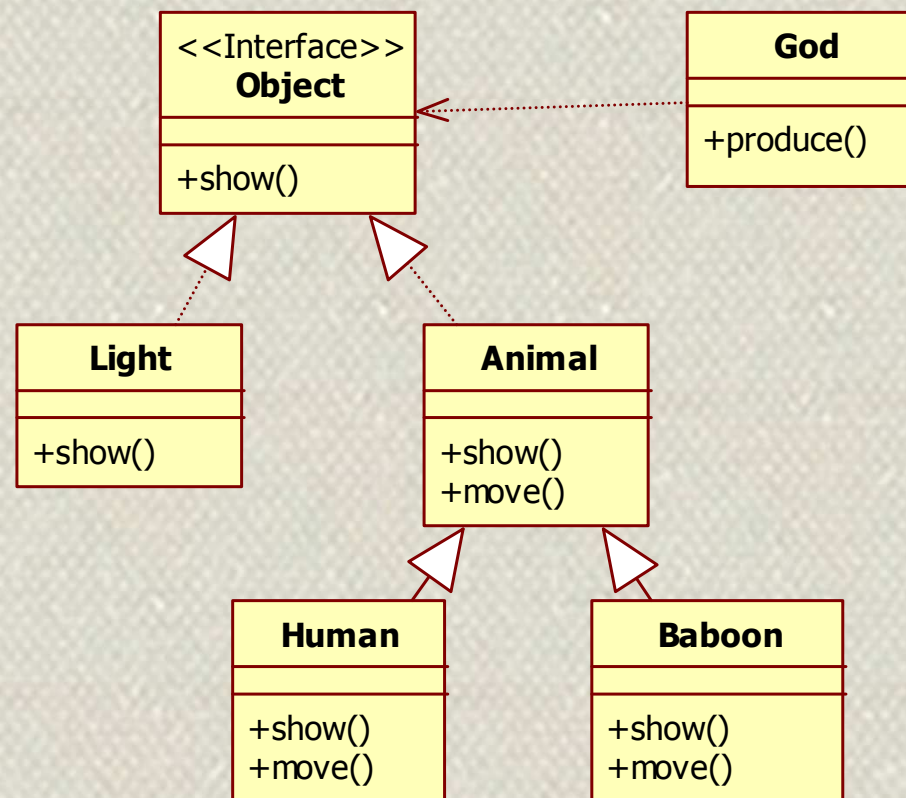
行为型

- 职责链 (Chain of Responsibility)
- 命令 (Command)
- 解释器 (Interpreter)
- 迭代器 (Iterator)
- 中介者 (Mediator)
- 备忘录 (Memento)
- 观察者 (Observer)
- 状态 (State)
- 策略 (Strategy)
- 模板方法 (Template Method)
- 访问者 (Visitor)

简单工厂

- 用统一的函数（方法）去创建对象
- 而不是随用随创建
- 生产主体是一个函数（方法）
- 要增加新产品非常不便，严格意义上不允许

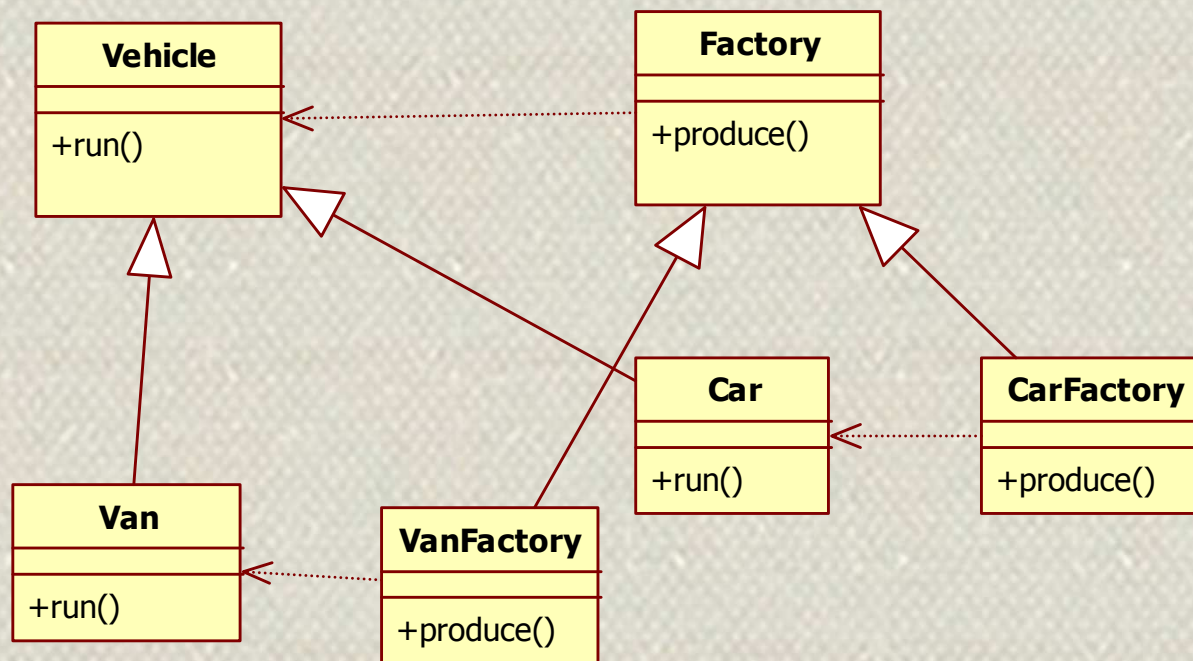
示例



工厂方法

- 工厂不是万能的，只能生产特定产品
- 小汽车工厂生产小汽车，巴士工厂生产巴士，坦克厂生产坦克
- 可以方便增加产品，增加产品，就要增加一个工厂

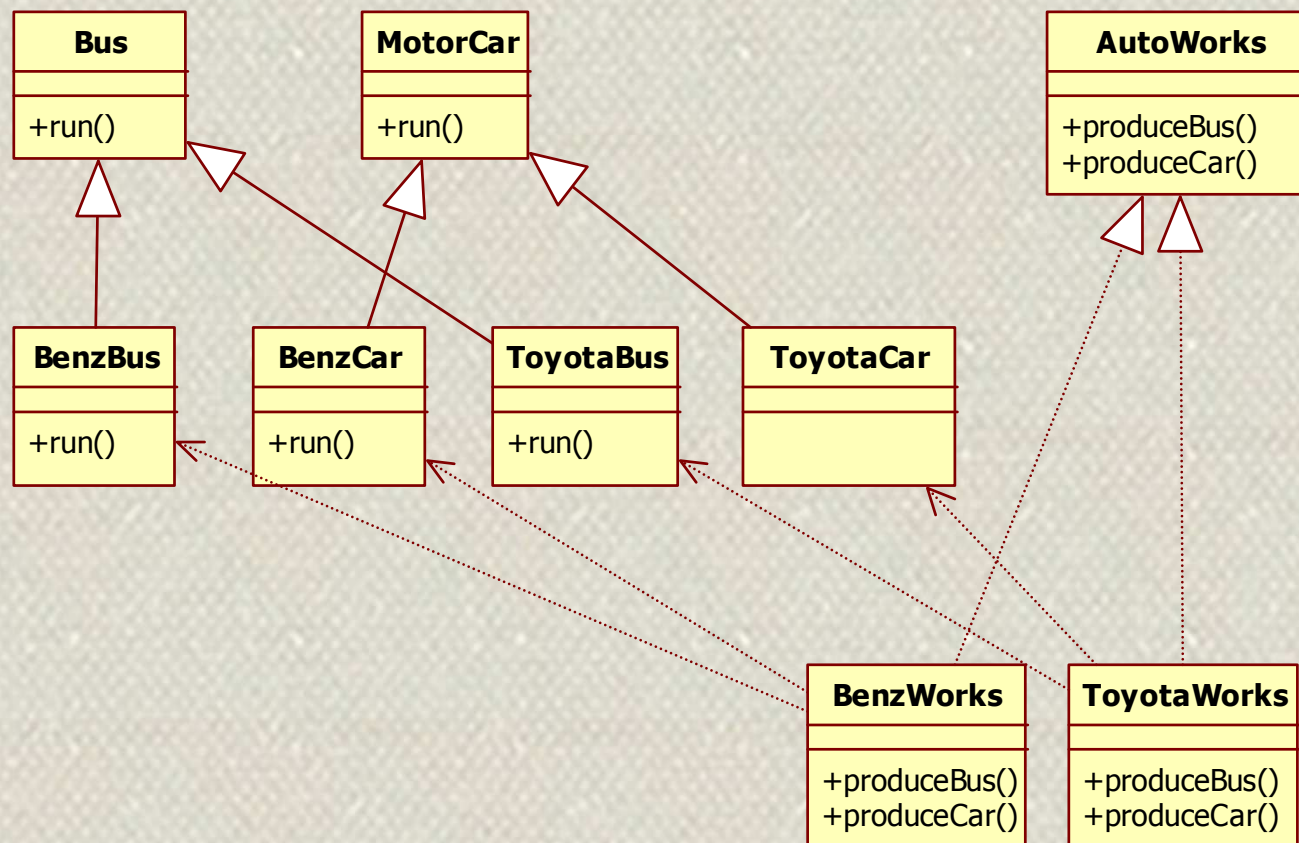
工厂方法



抽象工厂

- 汽车厂会生产多种汽车，大巴/轿车
- 各品牌汽车都有各自的汽车厂
- 方便添加汽车种类和汽车厂

抽象工厂



单例

- 一个类唯一的 1 个对象
- 一般用来存储全局的数据

单例模式

- 懒汉模式：使用时再实例化
- 饿汉模式：程序开始就实例化，不管之会不会用到

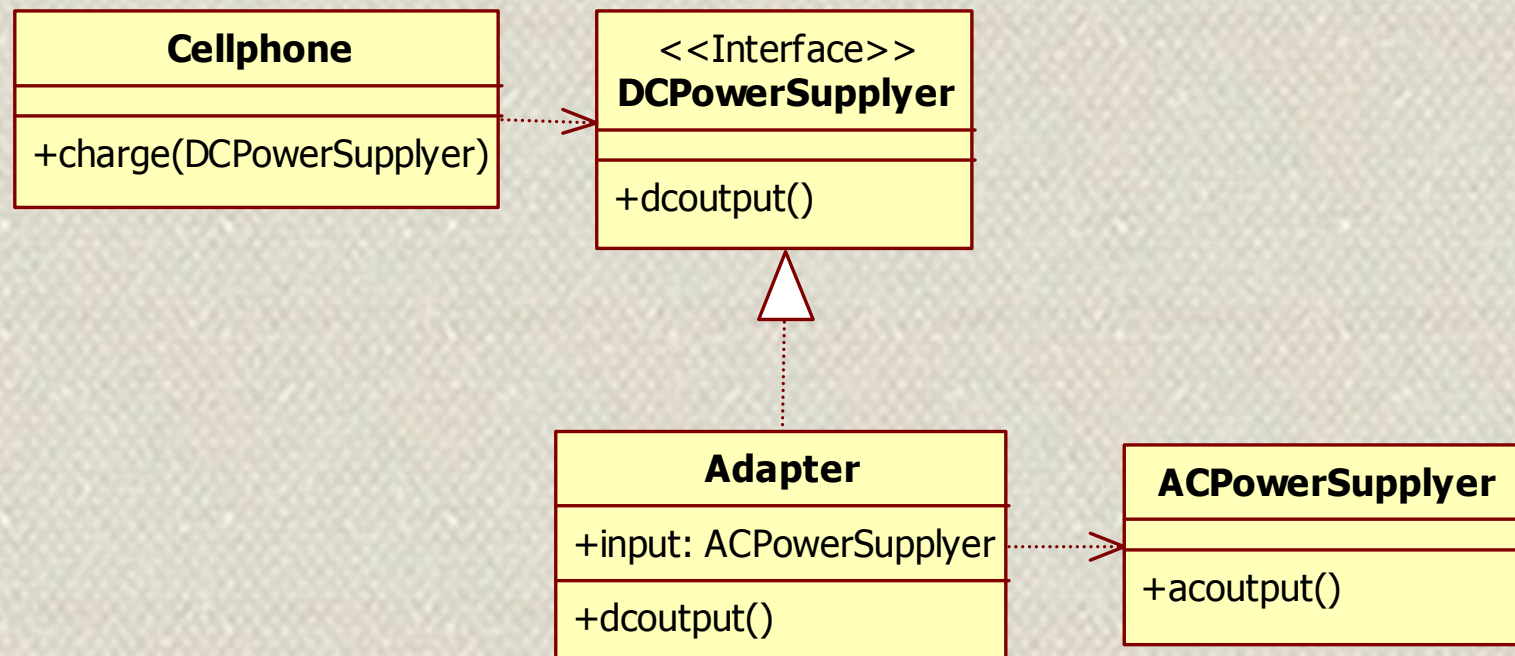
结构型

- 适配器(Adapter)
- 桥接(Bridge)
- 组合(Composite)
- 装饰(Decorator)
- 外观(Facade)
- 享元(Flyweight)
- 代理(Proxy)

适配器模式

- 系统需要使用现有的类，而这些类的接口不符合系统的需要
- 想要建立一个可以重复使用的类，用于与一些彼此之间没有太大关联的一些类，包括一些可能在将来引进的类一起工作
- 需要一个统一的输出接口，而输入端的类型不可预知

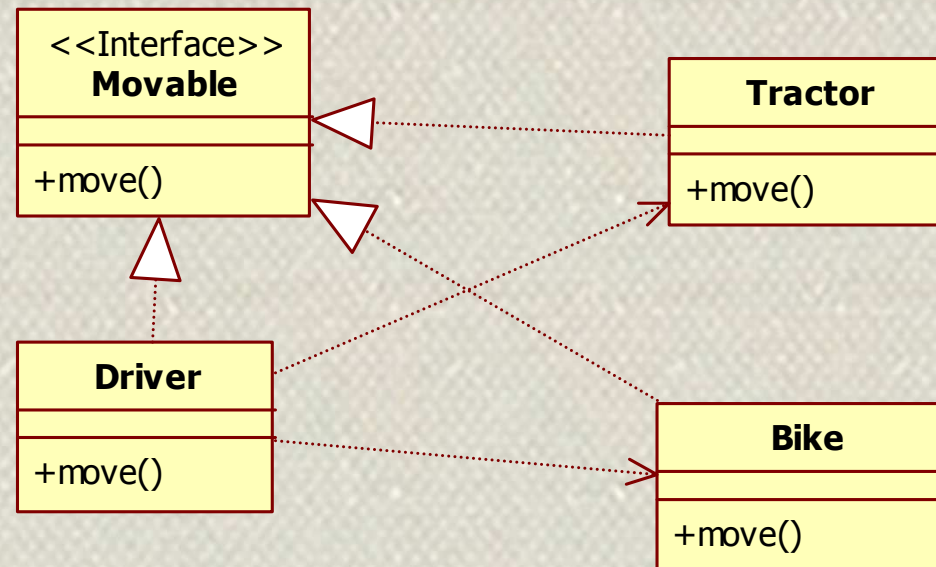
适配器模式



代理模式

- 用一个类代表另一个类的功能

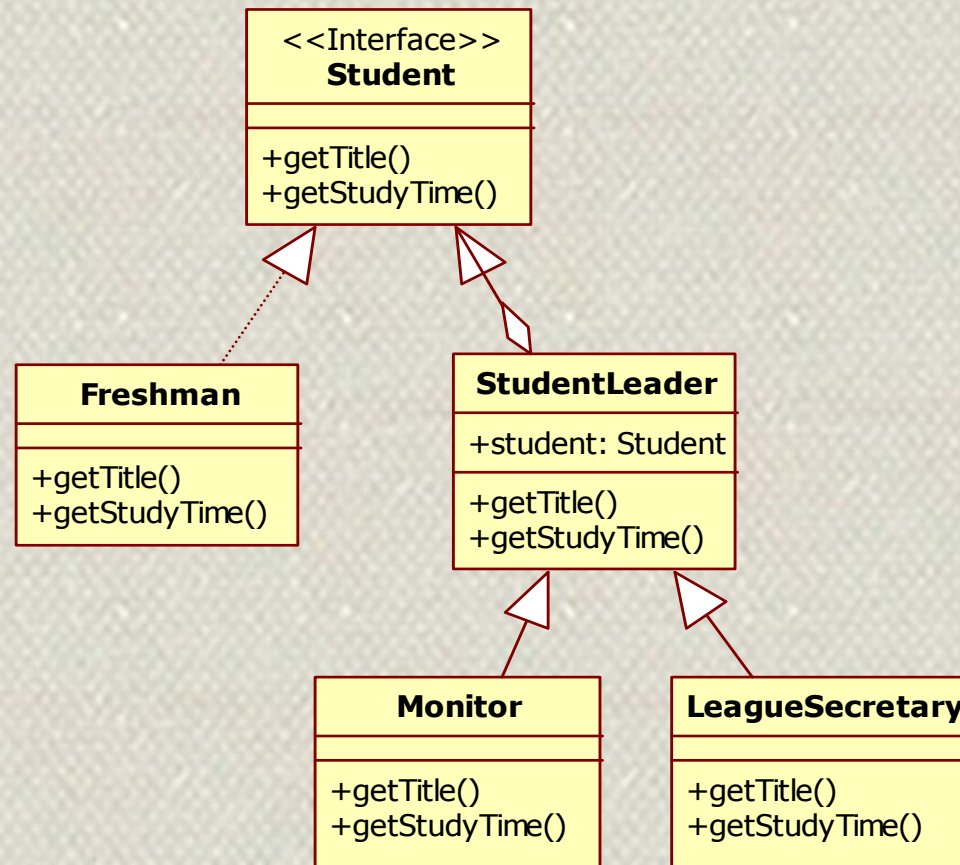
代理模式



装饰器模式

- 需要扩展一个类的功能，或给一个类增加附加责任
- 需要动态的给一个对象增加功能
- 需要增加一些基本功能的排列组合而产生的非常大量的功能，从而使继承变得不现实

装饰



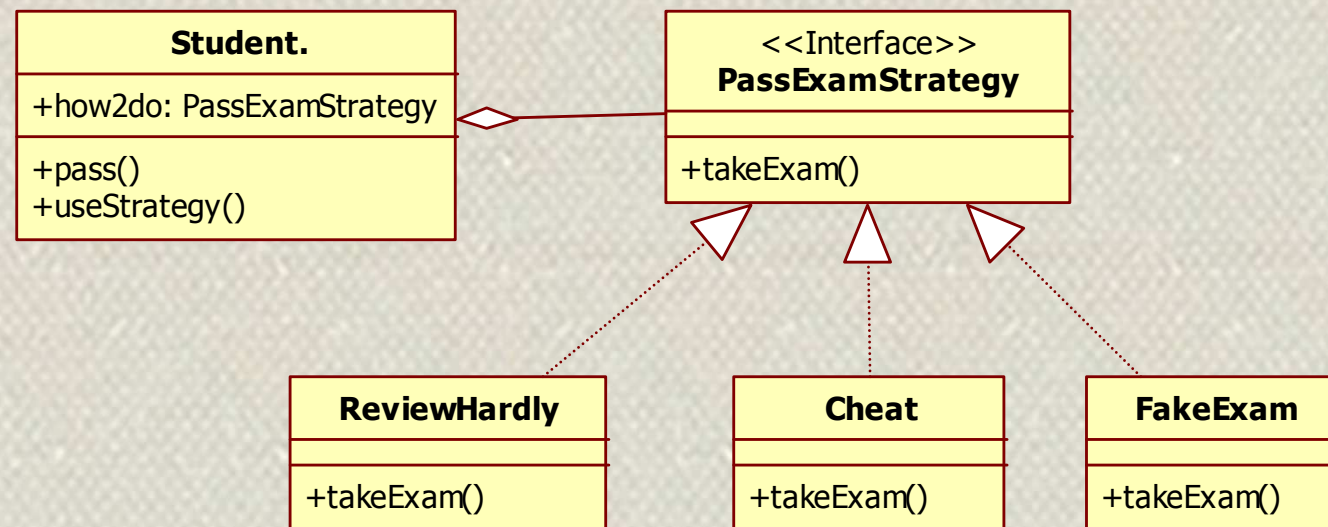
行为型

- 职责链 (Chain of Responsibility)
- 命令 (Command)
- 解释器 (Interpreter)
- 迭代器 (Iterator)
- 中介者 (Mediator)
- 备忘录 (Memento)
- 观察者 (Observer)
- 状态 (State)
- 策略 (Strategy)
- 模板方法 (Template Method)
- 访问者 (Visitor)

策略模式

- 策略模式是指对一系列的算法定义，并将每一个算法封装起来，而且使它们还可以相互替换
- 策略模式让算法独立于使用它的客户而独立变化。

策略模式



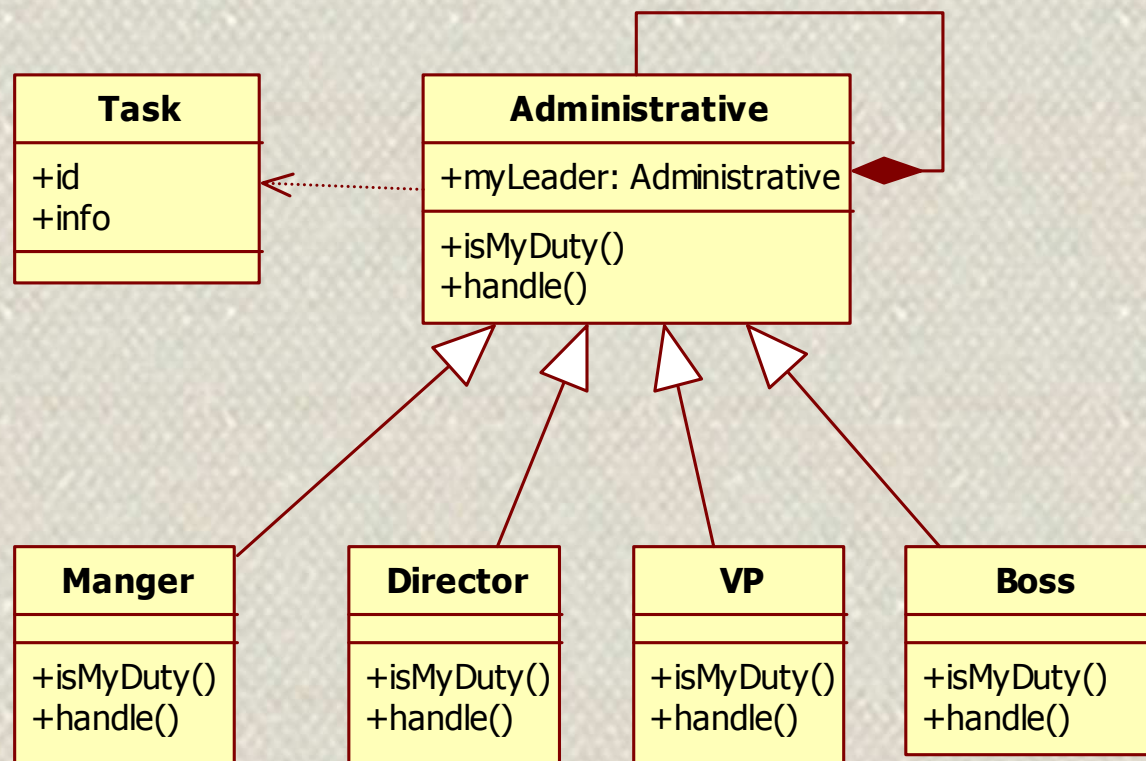
代理VS策略

- 代理和被代理，源自同一个基类，代替具体的职能
- 策略模式主体和使用之间不是同一个继承体系，根据不同的策略完成任务

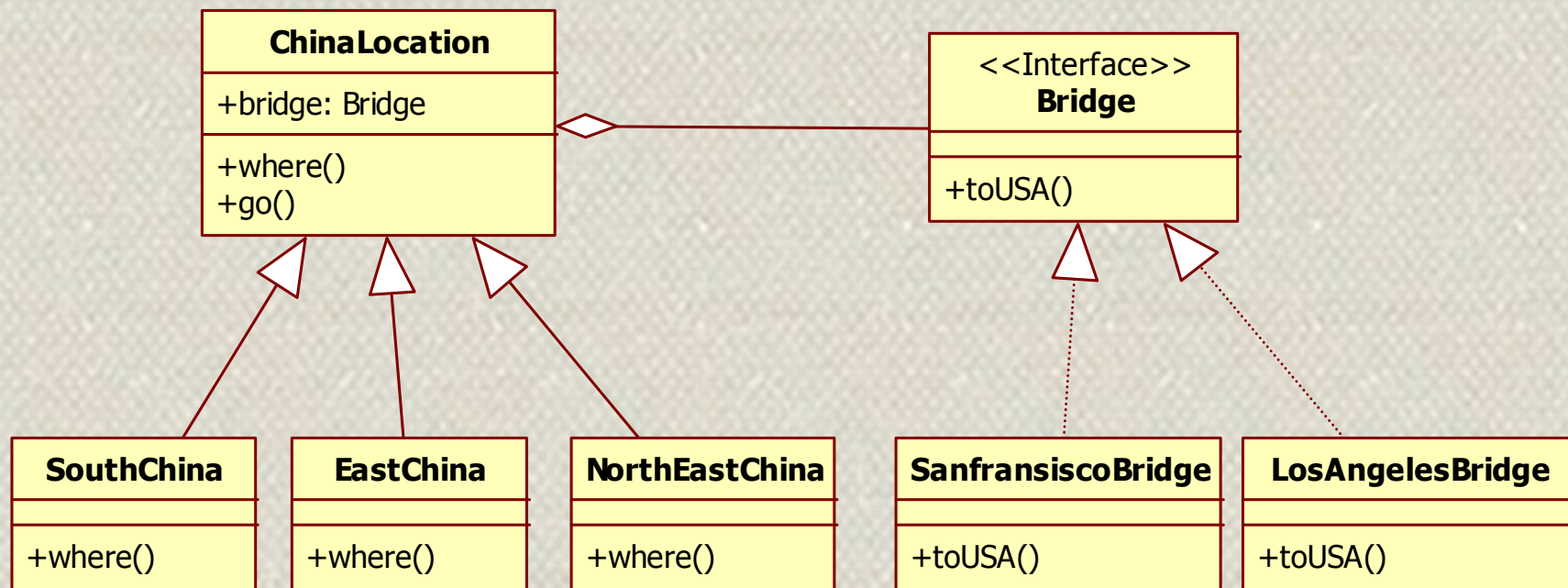
观察者模式

- 类图？

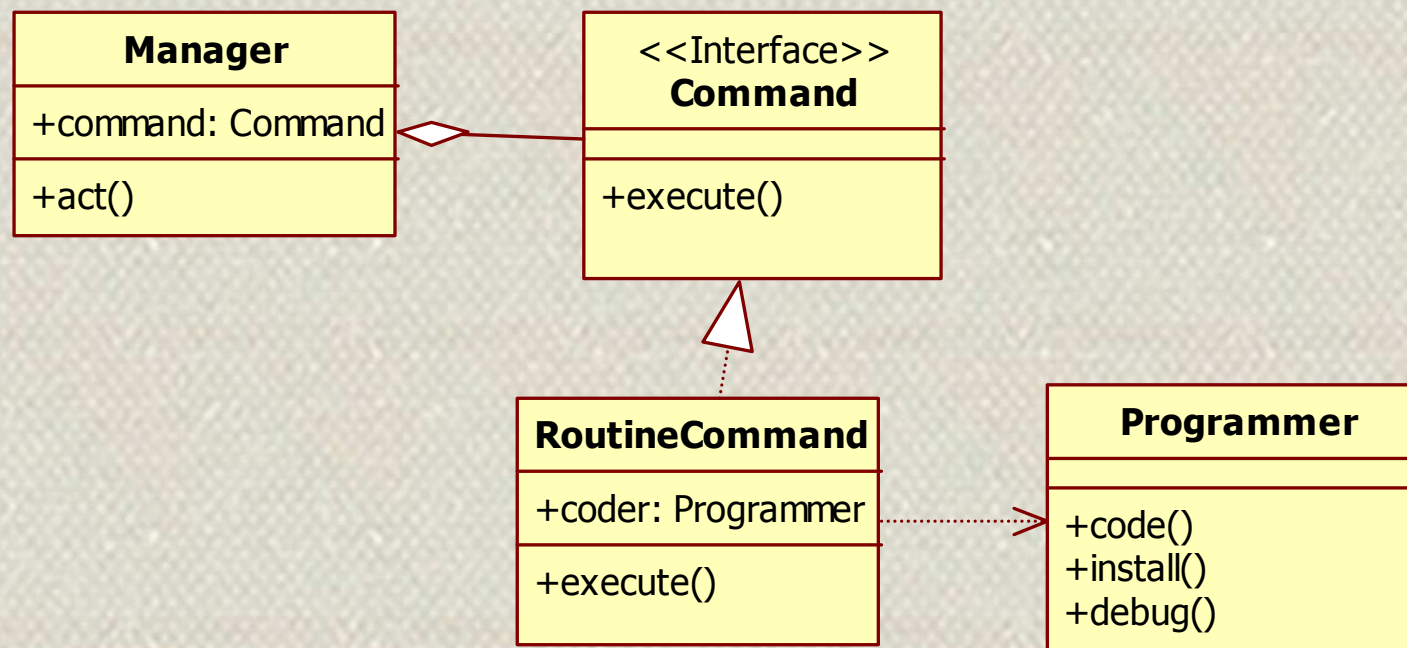
责任链



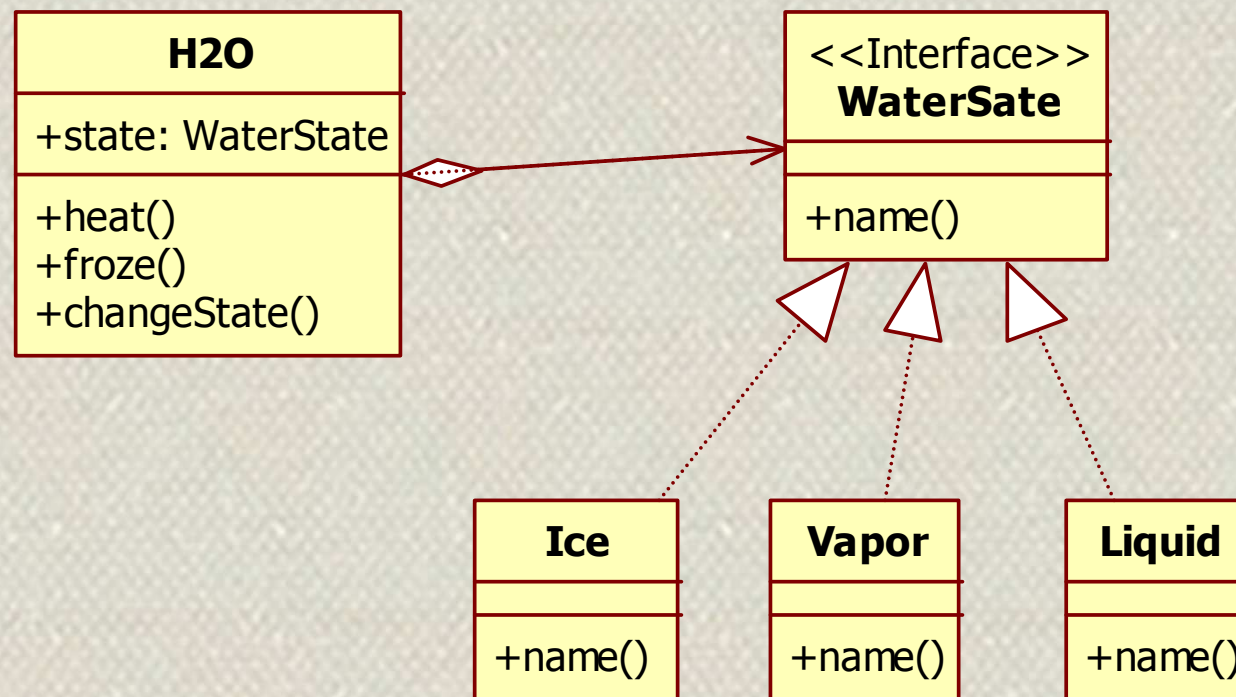
桥接



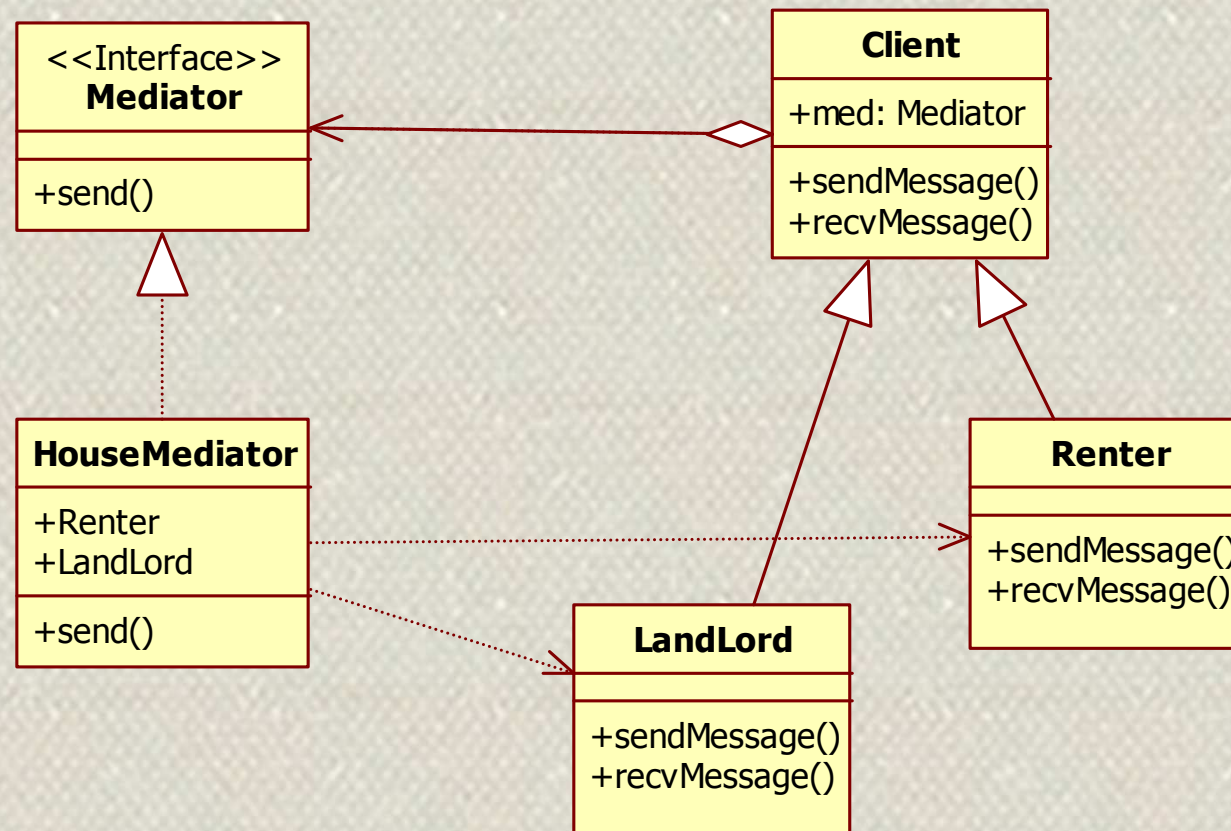
命令



状态



中介者



迭代器

