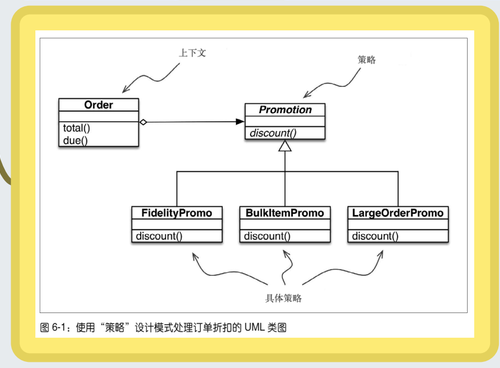
- 使用一等函数实现设计模式
  - 案例分析：重构“策略”模式
    - 如果合理利用作为一等对象的函数，某些设计模式可以简化，“策略”模式就是其中一个很好的例子。
    - 经典的“策略”模式
      - 定义一系列算法，把它们一一封装起来，并且使它们可以相互替换。本模式使得算法可以独立于使用它的客户而变化。
      - 上下文
        - 把一些计算委托给实现不同算法的可互换组件，它提供服务。
      - 策略
        - 实现不同算法的组件共同的接口。
      - 具体策略
        - “策略”的具体子类。
      - 

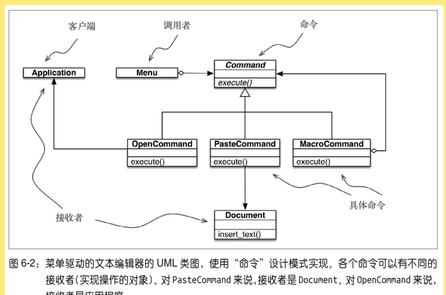
        图 6-1：使用“策略”设计模式处理订单折扣的 UML 类图
    - 使用函数实现“策略”模式
      - “策略对象通常是很好的享元（flyweight）。”
        - 享元是可共享的对象，可以同时在多个上下文中使用。
        - 共享是推荐的做法，这样不必在每个新的上下文（这里是 Order 实例）中使用相同的策略时不断新建具体策略对象，从而减少消耗。
      - 复杂的情况下，需要具体策略维护内部状态时，可能需要把“策略”和“享元”模式结合起来。但是，具体策略一般没有内部状态，只是处理上下文中的数据。
        - 一定要使用普通的函数，别去编写只有一个方法的类，再去实现另一个类声明的单函数接口。
      - 普通的函数也是“可共享的对象，可以同时在多个上下文中使用”。
    - 选择最佳策略：简单的方式
      - 数据结构存储函数
      - 可用，而且易于阅读，但是有些重复可能会导致不易察觉的缺陷：若想添加新的促销策略，要定义相应的函数
    - 找出模块中的全部策略
      - 在 Python 中，模块也是一等对象，而且标准库提供了几个处理模块的函数。
      - 内置函数 globals
        - 返回一个字典，表示当前的全局符号表。这个符号表始终针对当前模块（对函数或方法来说，是指定义它们的模块，而不是调用它们的模块）。
      - 高阶内省函数的 inspect 模块
        - inspect.getmembers 函数用于获取对象（这里是 promotions 模块）的属性，第二个参数是可选的判断条件（一个布尔值函数）。
        - inspect.isfunction，只获取模块中的函数。
  - “命令”模式
    - “命令”设计模式也可以通过把函数作为参数传递而简化。
    - 

      图 6-2：简单层级的文本编辑器的 UML 类图，使用“命令”设计模式实现。各个命令可以有不同的接收者（实现操作的对象）。对 PasteCommand 来说，接收者是 Document。对 OpenCommand 来说，接收者是应用程序
    - “命令”模式的目的是解耦调用操作的对象（调用者）和提供实现的对象（接收者）。
    - 这个模式的做法是，在二者之间放一个 Command 对象，让它实现只有一个方法（execute）的接口，调用接收者中的方法执行所需的操作。
      - 调用者无需了解接收者的接口
      - 而且不同的接收者可以适应不同的 Command 子类。
    - “命令模式是回调机制的面向对象替代品。”
      - 但并非始终需要回调机制的面向对象替代品
    - 实现成定义了 __call__ 方法的类。这样，MacroCommand 的实例就是可调用对象，各自维护着一个函数列表，供以后调用
      - 每个 Python 可调用对象都实现了单方法接口，这个方法就是 __call__。
    - 采用的方式与“策略”模式所用的类似：把实现单方法接口的类的实例替换成可调用对象。
  - 两个设计原则：“对接口编程，而不是对实现编程”和“优先使用对象组合，而不是类继承”。