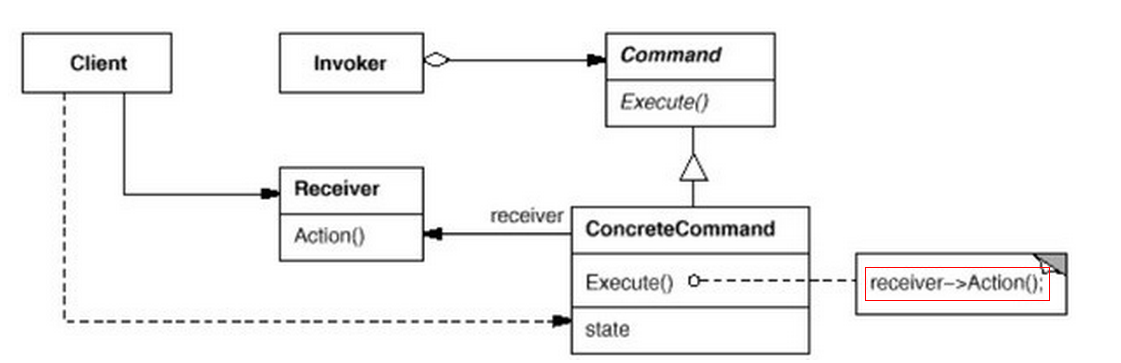
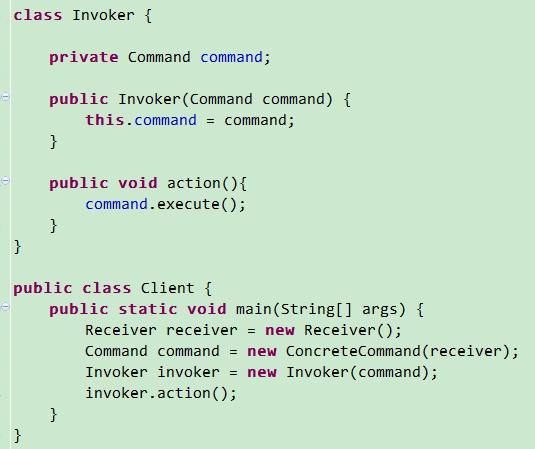
1. 命令（Command）模式：又称Action模式或者Transaction模式。在软件系统中，“行为请求者”与“行为实现者”通常呈现一种“紧耦合”。但在某些场合，比如要对行为进行“记录、撤销/重做、事务”等处理，这种无法抵御变化的紧耦合是不合适的。在这种情况下，如何将“行为请求者”与“行为实现者”解耦？将一组行为抽象为对象，实现二者之间的松耦合。这就是命令模式（Command Pattern）。
2. 类图



* 1. 抽象命令（Command）角色：定义命令的接口，声明执行的方法
  2. 具体命令（ConcreteCommand）角色：命令接口实现对象，是“虚”的实现；通常会持有接收者，并调用接收者的功能来完成命令要执行的操作。
  3. 客户端（Client）角色：创建具体的命令对象，并且设置命令对象的接收者。注意这个不是我们常规意义上的客户端，而是在组装命令对象和接收者，或许，把这个Client称为装配者会更好理解，因为真正使用命令的客户端是从Invoker来触发执行。
  4. 请求者（Invoker）角色：要求命令对象执行请求，通常会持有命令对象，可以持有很多的命令对象。这个是客户端真正触发命令并要求命令执行相应操作的地方，也就是说相当于使用命令对象的入口
  5. 接收者（Receiver）角色：接收者，真正执行命令的对象。任何类都可能成为一个接收者，只要它能够实现命令要求实现的相应功能。

1. 流程
   1. Client创建一个ConcreteCommand对象并指定他的Receiver对象
   2. 某个Invoker对象存储该ConcreteCommand对象
   3. 该Invoker通过调用Command对象的Execute操作来提交一个请求。若该命令是可撤销的，ConcreteCommand就在执行Execute操作之前存储当前状态以用于取消该命令
   4. ConcreteCommand对象对调用它的Receiver的一些操作以执行该请求
2. 例





1. 模式分析
   1. 命令模式的本质是对命令进行封装，将发出命令的责任和执行命令的责任分割开。
   2. 每一个命令都是一个操作：请求的一方发出请求，要求执行一个操作；接收的一方收到请求，并执行操作。
   3. 命令模式允许请求的一方和接收的一方独立开来，使得请求的一方不必知道接收请求的一方的接口，更不必知道请求是怎么被接收，以及操作是否被执行、何时被执行，以及是怎么被执行的。
   4. 命令模式使请求本身成为一个对象，这个对象和其他对象一样可以被存储和传递。
   5. 命令模式的关键在于引入了抽象命令接口，且发送者针对抽象命令接口编程，只有实现了抽象命令接口的具体命令才能与接收者相关联。
2. 开发中常见场景
   1. 数据库事务机制的底层实现
   2. 命令的撤销和恢复
3. 优点
   1. 降低对象之间的耦合度。
   2. 新的命令可以很容易地加入到系统中。
   3. 可以比较容易地设计一个组合命令。
   4. 调用同一方法实现不同的功能
4. 缺点
   1. 使用命令模式可能会导致某些系统有过多的具体命令类。因为针对每一个命令都需要设计一个具体命令类，因此某些系统可能需要大量具体命令类，这将影响命令模式的使用
5. 适用环境
   1. 系统需要将请求调用者和请求接收者解耦，使得调用者和接收者不直接交互。
   2. 系统需要在不同的时间指定请求、将请求排队和执行请求。
   3. 系统需要支持命令的撤销(Undo)操作和恢复(Redo)操作。
   4. 系统需要将一组操作组合在一起，即支持宏命令。