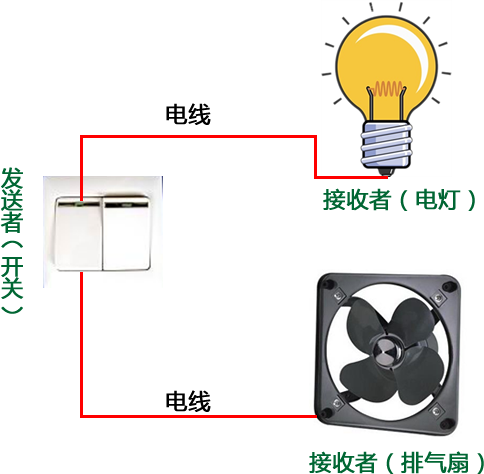
# 命令模式简介

命令模式将请求发送者与请求接收者解耦，请求发送者通过命令对象来间接引用接收者，使得系统具有更好的灵活性，可以在不修改现有系统源代码的情况下让相同的发送者对应不同的接收者。

## 概念

1. 在现实生活中人们通过开关来控制一些电器的打开和关闭，例如电灯或者排气扇，在购买开关时，人们并不知道它将来到底用于控制什么电器，也就是说开关与电灯、排气扇并不直接关系，一个开关在安装之后可能用来控制电灯，也可能用来控制排气扇或者其他电器设备。开关与电器之间通过电线建立连接，如果开关打开，则电线通电，电器工作；反之，开关关闭，电线断电，电器停止工作。相同的开关可以通过不同的电线来控制不同的电器。我们可以将开关理解成一个请求的发送者，用户通过它来发送一个“开灯”请求，而电灯是“开灯”请求的最终接收者和处理者，开关和电灯之间并不存在直接耦合关系，它们通过电线连接在一起，使用不同的电线可以连接不同的请求接收者，只需要更换一根电线，相关的发送者（开关）即可对应不同的接受者（电器）。如下图所示：



在软件开发中也存在很多与开关和电器类似的请求发送者和接收者对象，例如一个按钮，它可能是一个“关闭窗口”请求的发送者，而按钮单击事件处理类则是该请求的接收者。为了降低系统的耦合度，将请求的发送者和接受者解耦，可以使用一种被称为命令模式的设计模式来设计系统，在命令模式中发送者与接收者之间引入了新的命令对象（类似于上图的电线），将发送者的请求封装在命令对象中，再通过命令对象来调用接收者的方法。

1. 命令模式可以将请求发送者和接受者完全解耦，发送者与接受者之间没有直接引用关系，发送请求的对象只需要知道如何发送请求，而不必知道如何完成请求。
2. 命令模式的定义如下：命令模式（Command Pattern）将一个请求封装为一个对象，从而可用不同的请求对客户进行参数化，对请求排队或者记录请求日志，以及支持可撤销的操作。

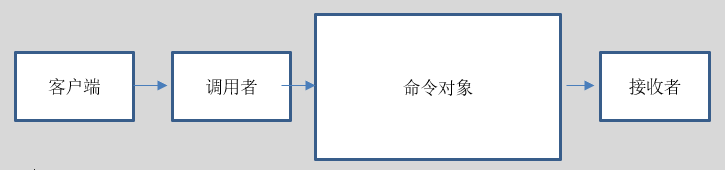
## 结构

命令模式的核心在于引入了抽象命令类和具体命令类，通过命令类来降低发送者和接受者的耦合度，请求发送者只需指定一个命令对象，再通过命令对象来调用请求接受者的处理方法。其结构图如下：



由图可知，命令模式包含以下4个角色：

1. Command（抽象命令类）：抽象命令类一般是一个抽象类或接口，在其中是声明了用于执行请求的execute( )等方法，通过这些方法可以调用请求接收者的相关操作。
2. ConcreteCommand（具体命令类）：具体命令类是抽象命令类的子类，并覆写了抽象命令类中声明的方法，它对应具体的接收者对象，将接收者对象的动作绑定其中。具体命令类在实现execute( )方法时将调用接收者对象的相关操作（Action）。
3. Invoker（调用者）：调用者即请求发送者，它通过命令对象来执行请求。一个调用者并不需要在设计时确定其接收者，因此它只与抽象命令类之间存在关联关系。在程序运行时可以将一个具体命令对象注入其中，再调用具体命令对象的execute( )方法，从而实现间接调用请求接收者的相关操作。
4. Receiver（接收者）：接收者执行与请求相关的操作，具体实现对请求的业务处理。



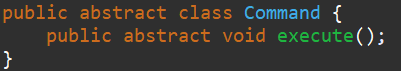
客户端针对调用者编程，在调用者内嵌了命令对象（相当于开关），通过使用不同的命令对象去调用不同的接受者。

# 实现

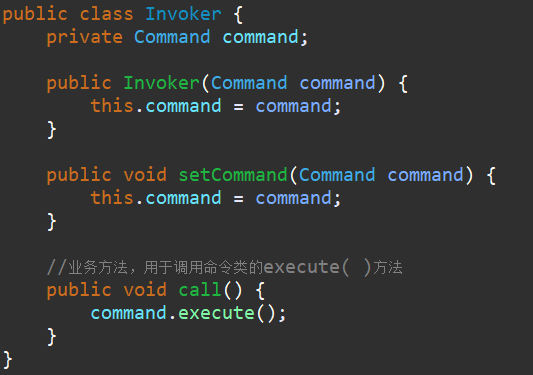
## 实现原理

命令模式的本质是对请求进行封装，一个请求对应于一个命令，将发出命令的责任和执行命令的责任分割开。每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相应的操作。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，更不必知道如何被接收，操作是否被执行、合适被执行，以及是怎么被执行的。

1. 命令模式的关键在于引入了抽象命令类，请求发送者针对抽象命令类编程，只有实现了抽象命令类的具体命令才与请求接受者相关联。在最简单的抽象命令类中只包含一个抽象的execute( )方法， 每个具体命令类将一个 Receiver类型的对象作为一个实例变量进行存储，从而具体指定一个请求的接受者，不同的具体命令类提供了execute( )方法的不同实现，并调用不同接收者的请求处理方法。典型的抽象命令类代码如下：



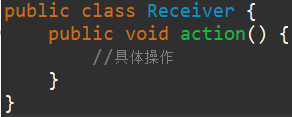
1. 对于请求发送者（即调用者）而言，将针对抽象命令类进行编程，可以通过调用构造函数或者setter方法在运行时注入具体命令类对象，并在业务方法中调用命令对象的execute( )方法，其典型代码如下：



1. 具体命令类继承了抽象命令类，它与请求接受者相关联，实现了在抽象命令类中声明的execute( )方法，并在实现时调用接收者的请求响应方法action( )，其典型代码如下：



1. 请求接收者Receiver具体实现对请求的业务处理，它拥有action( )方法，用于执行与请求相关的操作，其典型代码如下：

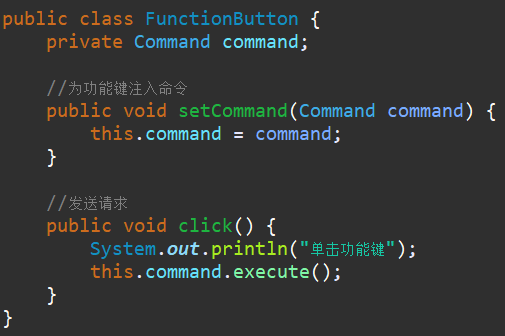


## 实例

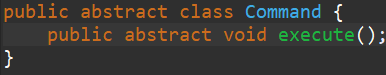
实例说明：为了用户使用方便，某系统提供了一系列功能键，用户可以自定义功能键的功能，例如功能键FunctionButton可以用于退出系统（由SystemExitClass类实现），也可以用于显示帮助文档（由DisplayHelpClass类实现）。用户可以通过修改配置文件改变功能键的用途，现使用命令模式设计该系统，使得功能键类与功能类之间解耦，可为同一个功能键设置不同的功能。 通过分析，本实例的结构图如下所示：



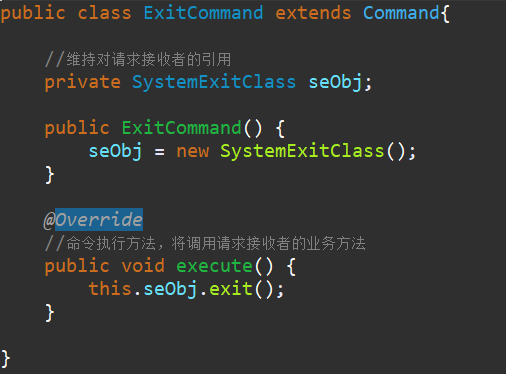
1. FunctionButton：功能键类，充当请求调用者（请求发送者）

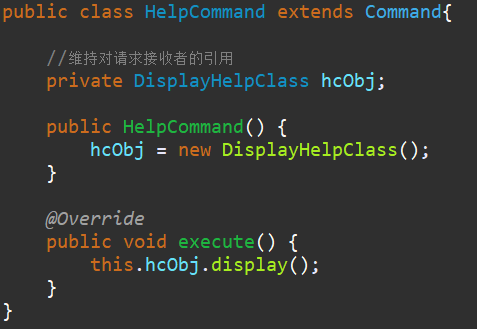


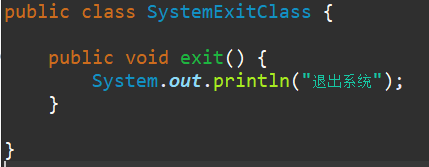
1. Command：充当抽象命令类



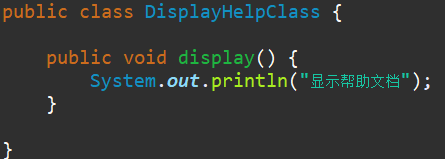
1. ExitCommand：退出命令类，抽象命令类的子类，充当具体命令类



1. HelpCommand：帮助命令类，抽象命令类的子类，充当具体命令类
2. SystemExitClass：退出系统模拟实训类，充当请求接受者

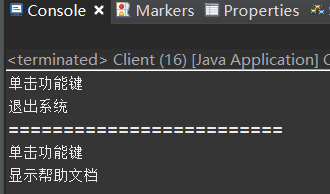


1. DisplayHelpClass：显示帮助文档模拟实训类，充当请求接受者



1. Client：客户端测试类，测试命令模式



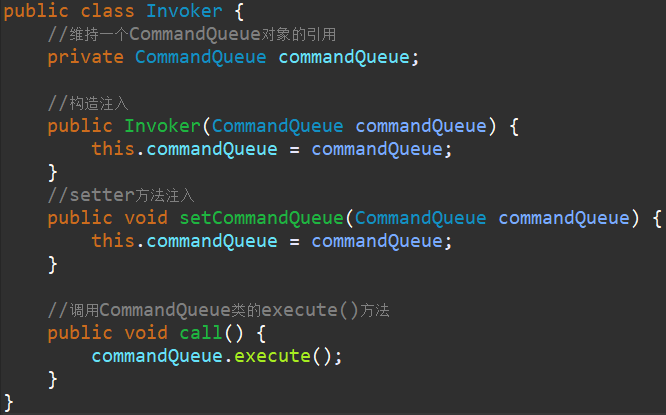
 运行结果如下：

## 实现命令队列

有时候，当一个请求发送（即调用者）者发送一个请求时有不止一个请求接受者（即接收者）产生响应，这些请求接受者（接收者）将逐个执行业务方法，完成对请求的处理，此时可以通过命令队列实现。命令队列的实现方法有很多种形式，其中最常用的、灵活性最好的一种方式是增加一个CommandQueue类，由该类负责存储多个命令对象，而不同的命令对象可以对应不同的请求接收者。CommandQueue类的典型代码如下：



在增加了命令队列类CommandQueue以后，请求发送者（调用者）Invoker将针对CommandQueue编程，其代码修改如下：



命令队列与人们常说的“批处理”有点类似。批处理，顾名思义，可以对一组命令对象进行批处理，当一个发送者发送请求后将有一系列接收者对请求作出响应，命令队列可以用于设计批处理应用程序，如果请求接受者的接收次序没有严格的先后次序，还可以使用多线程技术来并发调用命令对象的execute( )方法，从而提高程序的执行效率。

## 记录请求日志

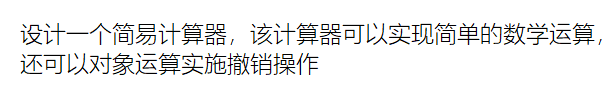
请求日志就是将请求的历史记录保存下来，通常以日志文件（Log File）的形式永久保存在计算机中。很多系统都提供了日志文件，例如Windows日志文件，Oracle日志文件等。日志文件可以记录用户对系统的一些操作（例如对数据的更改）。请求日志文件可以实现很多功能，常用功能如下：

1. 一旦系统发生故障，日志文件可以为系统提供一种恢复机制，在请求日志文件中可以记录用户对系统的每一步操作，从而让系统能够顺利恢复到某一个特定的状态。
2. 请求日志也可以用于实现批处理，在一个请求日志文件中可以存储一系列命令对象，例如一个命令队列。
3. 可以将命令队列中的所有命令对象都存储在一个日志文件中，每执行一个命令则从日志文件中删除一个对应的命令对象，防止因为断电或者系统重启等原因造成请求丢失，而且可以避免重新发送全部请求时造成某些命令的重复执行，只需读取请求日志文件，再继续执行文件中剩余的命令即可。

在实现请求日志时可以将发送请求的命令对象通过序列化写到日志文件中，此时命令类必须实现java.io.Serializable接口。

## 实现撤销操作

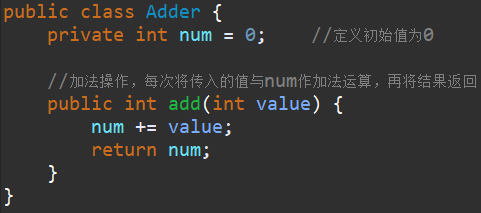
在命令模式红可以通过对命令类进行修改使得系统支持撤销（Undo）操作和恢复（Redo）操作，通过一个简单实例来学习如何在命令模式中实现撤销操作。



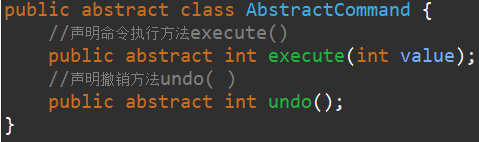
现使用命令模式设计可得到如下图所示的结构图：



1. 加法类Adder充当请求接受者（接受者），其代码如下：



1. AbstractCommand充当抽象命令类，声明了execute( )方法和撤销方法undo( )，其代码如下：



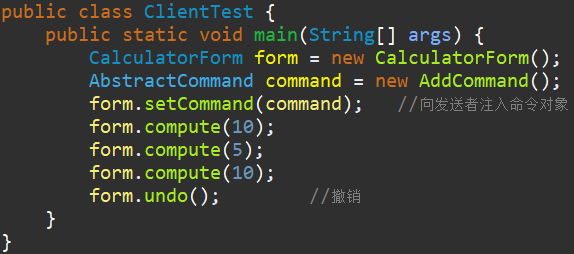
1. AddCommand充当具体命令类，实现了在抽象命令类AbstractCommand中声明的execute( )方法和撤销方法undo( )，其代码如下：

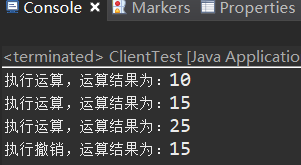


1. CalculatorForm充当请求发送者（即调用者），它引用一个抽象命令类型AbstractCommand的对象command，通过command对象间接调用请求接收者Adder类的业务处理方法，其代码如下：



1. 在客户端测试类Client中定义了抽象命令类型的命令对象command，还创建了请求发送者（即调用者）对象form，通过调用form对象的compute( )方法实现加法运算，还可以调用undo( )方法撤销最后一次加法运算，其代码如下：



 编译并运行程序，结果如下：

需要注意的是在本实例中只能实现一步撤销操作，因为没有保存命令对象的历史状态，可以通过引入一个命令集合或其他方式来存储每一次操作时的状态，从而实现多次撤销操作。除了Undo（撤销）操作外，还可以采用类似的方式实现恢复（Redo）操作，即恢复所撤销的操作（或称为二次撤销）。

## 宏命令

宏命令（Macro Command）又称为组合命令（Composite Command），它是组合模式和命令模式联用的产物。宏命令是一个具体命令类，它拥有一个集合，在该集合中包含了对其他命令对象的引用。通常宏命令不直接与请求接收者交互，而是通过它的成员来调用接收者的方法。当调用宏命令的execute( )方法时将递归调用它所包含的每个成员命令的execute( )方法，一个宏命令的成员可以是简单命令，也可以继续是宏命令。执行一个宏命令将触发多个具体命令的执行，从而实现对命令的批处理。其结构图如下：



# 优缺点及适用环境

命令模式是一种使用频率非常高的设计模式，它可以将请求发送者与接收者解耦，请求发送者通过命令对象来间接引用请求接收者，使得系统具有更好的灵活性和可扩展性。在基于GUI的软件开发（无论是电脑软件桌面应用还是手机移动应用）中，命令模式都得到了广泛的应用。

## 优点

命令模式的优点主要如下：

1. 降低系统的耦合度。由于请求者与接收者之间不存在直接引用，因此请求者与接收者之间实现完全解耦，相同的请求者可以对应不同的接收者，同样相同的接收者也可以供不同的请求者使用，两者之间具有良好的独立性。
2. 新的命令可以很容易地加入到系统汇总。由于增加新的具体命令类不会影响到其他类，因此增加新的具体命令类很容易，无须修改原有系统源代码，甚至客户端类代码，满足开闭原则的要求。
3. 可以比较容易地设计一个命令队列或宏命令（组合命令）。
4. 为请求的撤销（Undo）和恢复（Redo）操作提供了一种设计和实现方案。

## 缺点

命令模式的缺点主要如下：使用命令模式可能会导致某些系统有过多的具体命令类。因为针对每一个对请求接收者的调用操作都需要设计一个具体命令类，所以在某些系统中可能需要提供大量的具体命令类，这将影响命令模式的使用。

## 适用环境

在以下情况下可以考虑使用命令模式：

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。请求调用者无须知道接收者的存在，也无须知道接收者是谁，接收者也无须关心何时被调用。
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。一个命令对象和请求的初始调用者可以有不同的生命期，换而言之，最初的请求发出者可能已经不存在了，而命令对象本身仍然是活动的，可以通过该命令对象去调用请求接收者，并且无须关心请求调用者的存在性，可以通过请求日志文件等机制来具体实现。
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形成宏命令。