



深圳大学
Shenzhen University

第14-1讲

命令模式

软件体系结构与设计模式
Software Architecture & Design Pattern

深圳大学计算机与软件学院

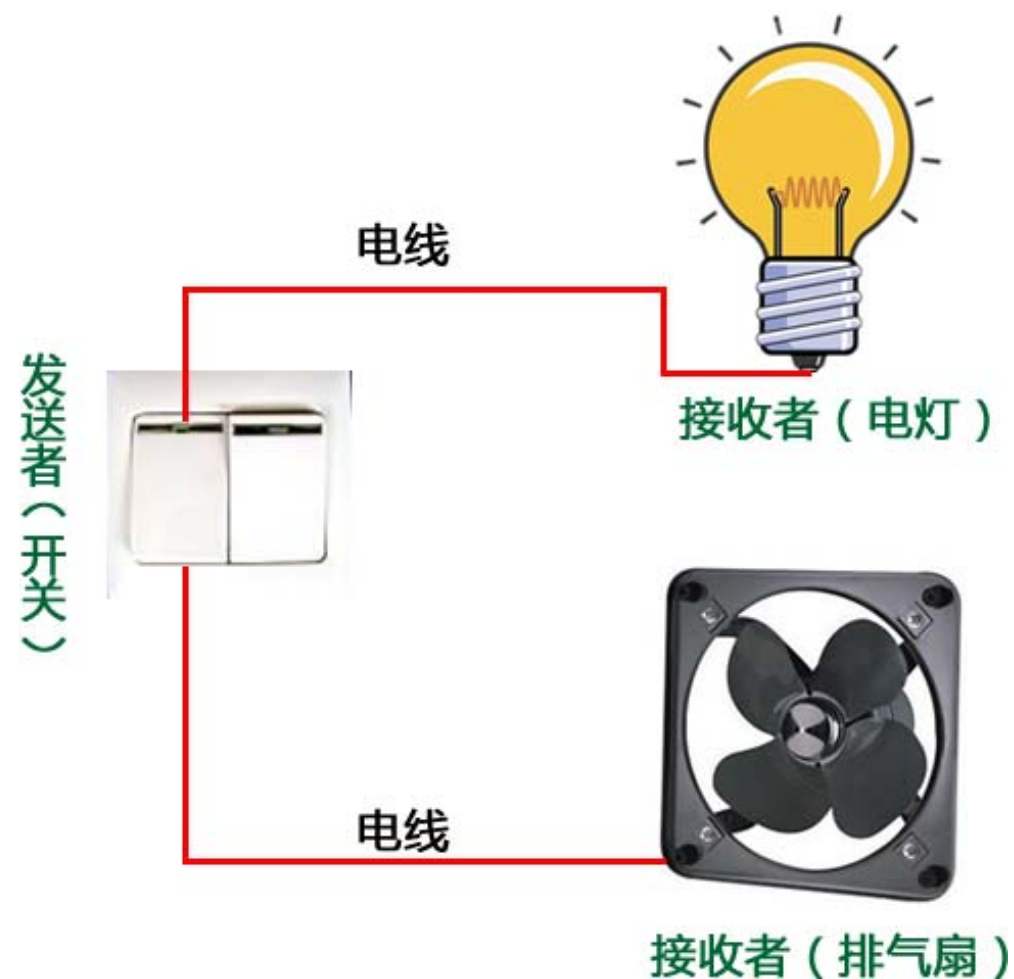


主要内容

- ◆ 命令模式动机与定义
- ◆ 命令模式结构与分析
- ◆ 命令模式实例与解析
- ◆ 命令模式效果与应用

命令模式动机

■ 开关与电灯、排气扇示意图





命令模式动机

■ 现实生活

- 相同的开关可以通过不同的电线来控制不同的电器
- 开关 $\leftarrow \rightarrow$ 请求发送者
- 电灯 $\leftarrow \rightarrow$ 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



命令模式动机

■ 软件开发

- 按钮 $\leftarrow \rightarrow$ 请求发送者
- 事件处理类 $\leftarrow \rightarrow$ 请求的最终接收者和处理者
- 发送者与接收者之间引入了新的命令对象（类似电线）
，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
- 相同的按钮可以对应不同的事件处理类



命令模式定义

- 对象行为型模式

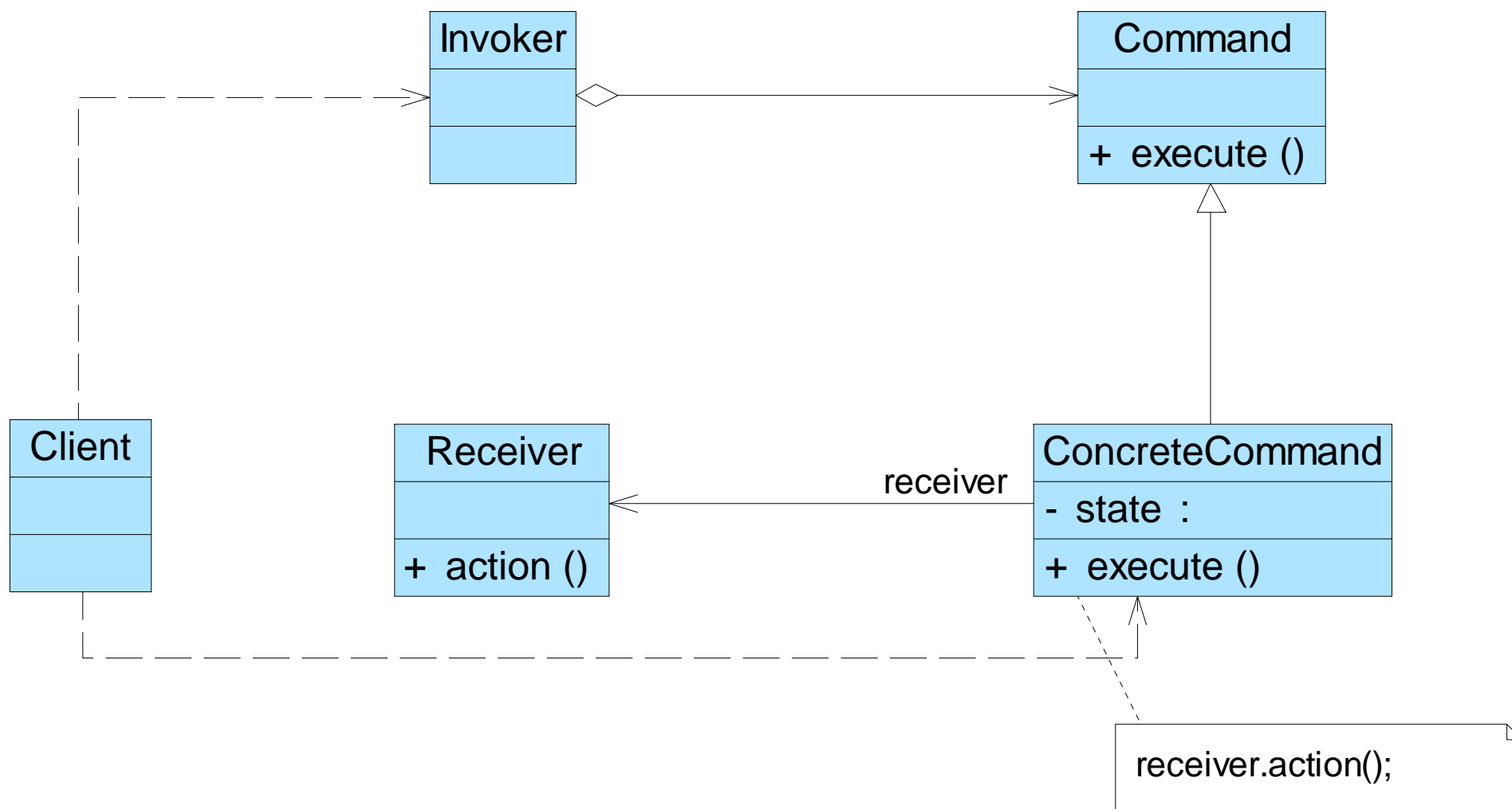
命令模式： 将一个请求封装为一个对象，从而让你可以用不同的请求对客户进行参数化，对请求排队或者记录请求日志，以及支持可撤销的操作。

Command Pattern: Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations.

- 别名为动作(Action)模式或事务(Transaction)模式

命令模式结构

- 命令模式包含如下角色：**Command**: 抽象命令类，
- **ConcreteCommand**: 具体命令类，**Invoker**: 调用者，**Receiver**: 接收者





命令模式分析

- 将请求发送者和接收者完全解耦
- 发送者与接收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
- 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开



命令模式分析

■ 调用者（请求发送者）示例代码：

```
public class Invoker {  
    private Command command;  
  
    //构造注入  
    public Invoker(Command command) {  
        this.command = command;  
    }  
  
    //设值注入  
    public void setCommand(Command command) {  
        this.command = command;  
    }  
  
    //业务方法，用于调用命令类的execute()方法  
    public void call() {  
        command.execute();  
    }  
}
```



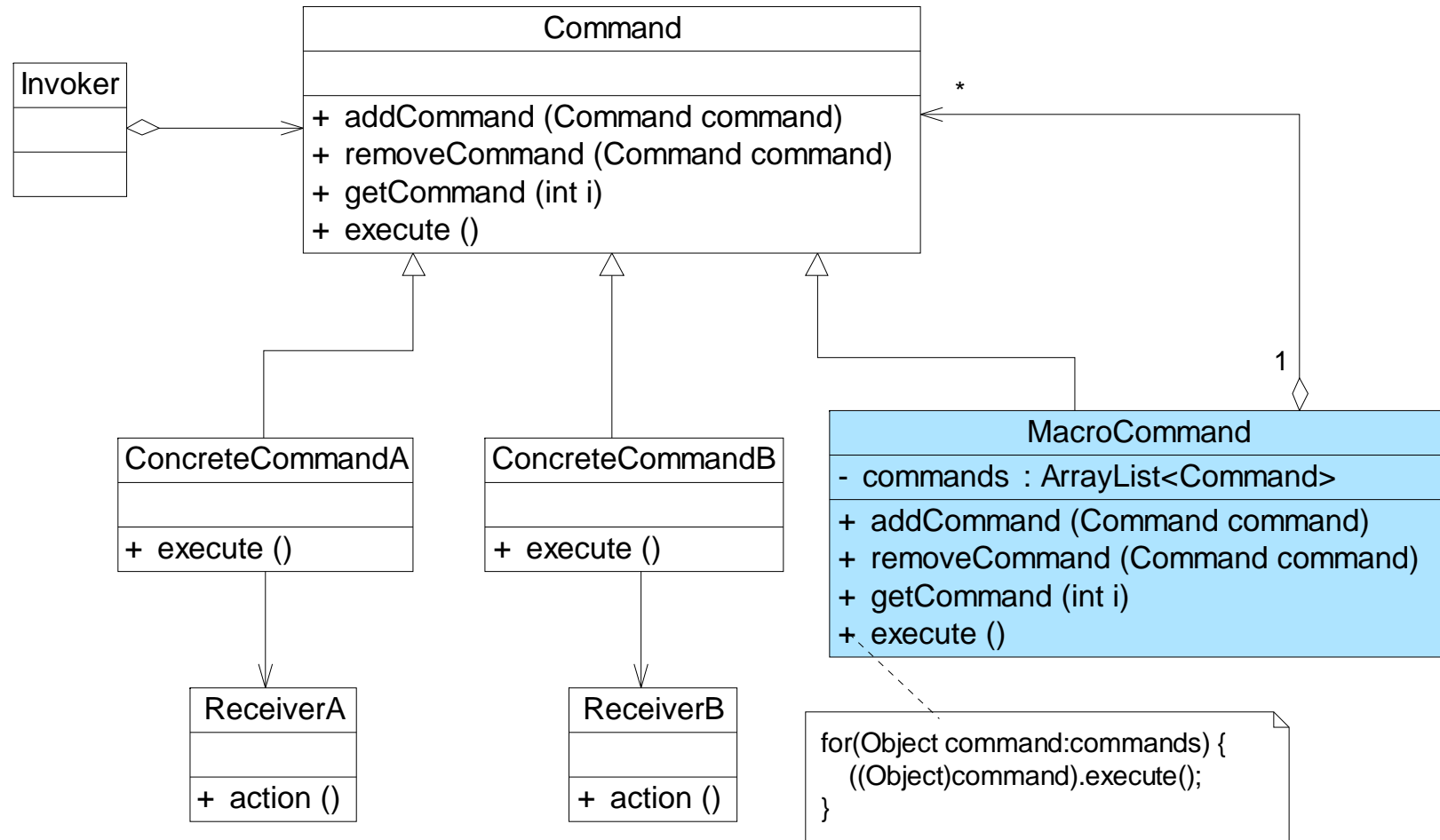
命令模式分析

■ 具体命令类示例代码：

```
public class ConcreteCommand extends Command {  
    private Receiver receiver; //维持一个对请求接收者对象的引用  
  
    public void execute() {  
        receiver.action(); //调用请求接收者的业务处理方法action()  
    }  
}
```

命令模式分析

■ 宏命令





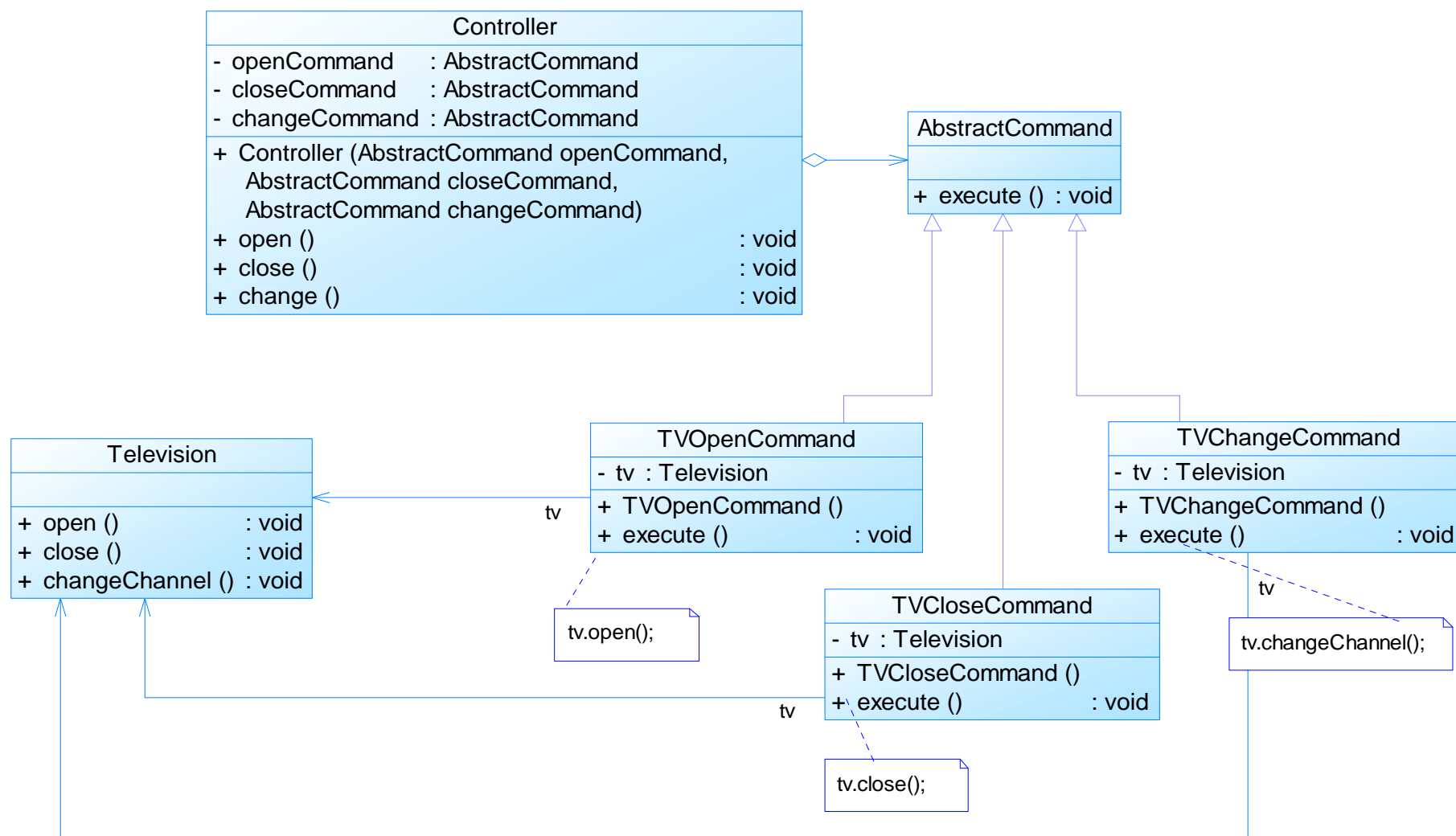
命令模式实例

■ 电视机遥控器：实例说明

- 电视机是请求的接收者，遥控器是请求的发送者，遥控器上有一些按钮，不同的按钮对应电视机的不同操作。抽象命令角色由一个命令接口来扮演，有三个具体的命令类实现了抽象命令接口，这三个具体命令类分别代表三种操作：打开电视机、关闭电视机和切换频道。显然，电视机遥控器就是一个典型的命令模式应用实例。

命令模式实例与解析

■ 电视机遥控器：参考类图



命令模式实例与解析

■ 命令模式实例

□ 电视机遥控器：参考代码

■ DesignPatterns之command包

```
AbstractCommand.java ✖  
1 package command;  
2  
3 public interface AbstractCommand  
4 {  
5     public void execute();  
6 }
```

Controller.java

```
3 public class Controller
4 {
5     private AbstractCommand openCommand,closeCommand,changeCommand;
6
7     public Controller(AbstractCommand openCommand,AbstractCommand
8         closeCommand,AbstractCommand changeCommand)
9     {
10         this.openCommand=openCommand;
11         this.closeCommand=closeCommand;
12         this.changeCommand=changeCommand;
13     }
14
15     public void open()
16     {
17         openCommand.execute();
18     }
19
20     public void change()
21     {
22         changeCommand.execute();
23     }
24
25     public void close()
26     {
27         closeCommand.execute();
28     }
29 }
```

Television.java ✕

```
1 package command;
2
3 public class Television
4 {
5     public void open()
6     {
7         System.out.println("打开电视机！");
8     }
9
10    public void close()
11    {
12        System.out.println("关闭电视机！");
13    }
14
15    public void changeChannel()
16    {
17        System.out.println("切换电视频道！");
18    }
19 }
```


TVOpenCommand.java

```
3 public class TVOpenCommand implements AbstractCommand
4 {
5     private Television tv;
6     public TVOpenCommand()
7     {
8         tv = new Television();
9     }
10    public void execute()
11    {
12        tv.open();
13    }
14 }
```

TVCloseCommand.java

```
3 public class TVCloseCommand implements AbstractCommand
4 {
5     private Television tv;
6     public TVCloseCommand()
7     {
8         tv = new Television();
9     }
10    public void execute()
11    {
12        tv.close();
13    }
14 }
```

TVChangeCommand.java ✕

```
1 package command;
2
3 public class TVChangeCommand implements AbstractCommand
4 {
5     private Television tv;
6     public TVChangeCommand()
7     {
8         tv = new Television();
9     }
10    public void execute()
11    {
12        tv.changeChannel();
13    }
14 }
```

Client.java

```
1 package command;
2
3 public class Client
4 {
5     public static void main(String a[])
6     {
7         AbstractCommand openCommand, closeCommand, changeCommand;
8
9         openCommand = new TVOpenCommand();
10        closeCommand = new TVCloseCommand();
11        changeCommand = new TVChangeCommand();
12
13        Controller control = new Controller(openCommand,
14                                           closeCommand, changeCommand);
15
16        control.open();
17        control.change();
18        control.close();
19    }
20 }
```



命令模式效果与应用

■ 命令模式优点：

- 降低系统的耦合度
- 新的命令可以很容易地加入到系统中，符合开闭原则
- 可以比较容易地设计一个命令队列或宏命令（组合命令）
- 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



命令模式效果与应用

■ 命令模式缺点：

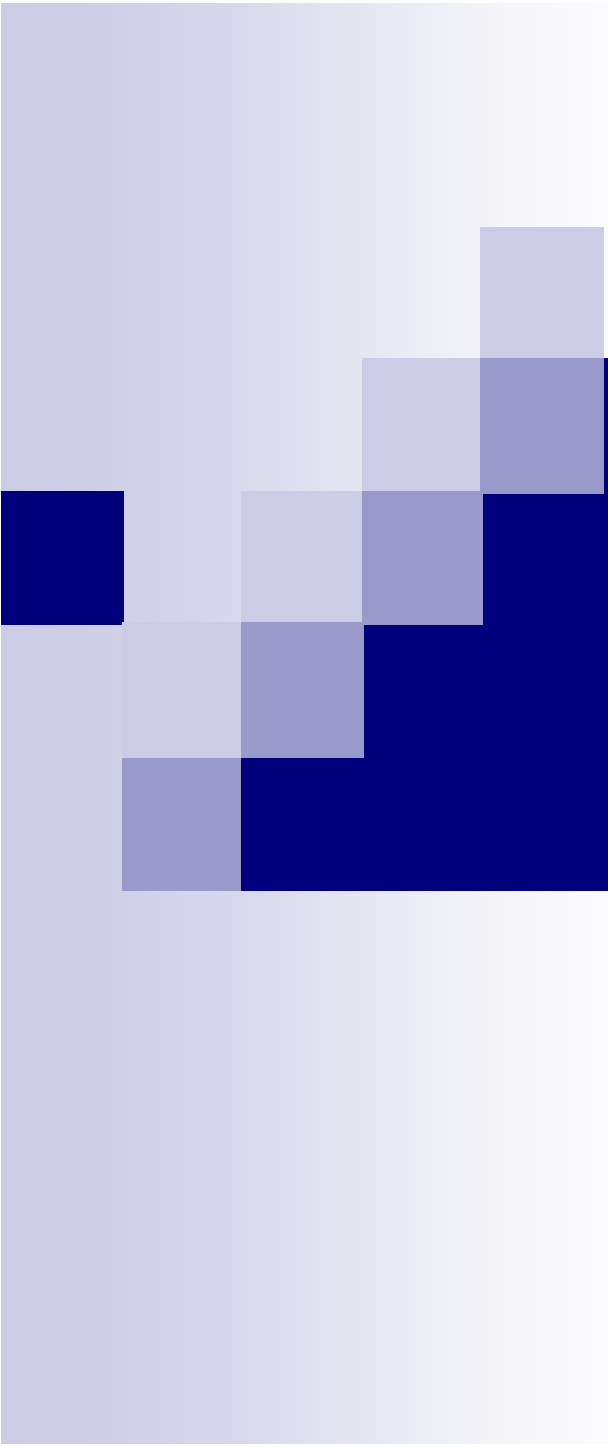
- 使用命令模式可能会导致某些系统有过多的具体命令类（针对每一个对请求接收者的调用操作都需要设计一个具体命令类）



命令模式效果与应用

■ 在以下情况下可以使用命令模式：

- 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
- 需要在不同的时间指定请求、将请求排队和执行请求
- 需要支持命令的撤销(Undo)操作和恢复(Redo)操作
- 需要将一组操作组合在一起形成宏命令



谢谢