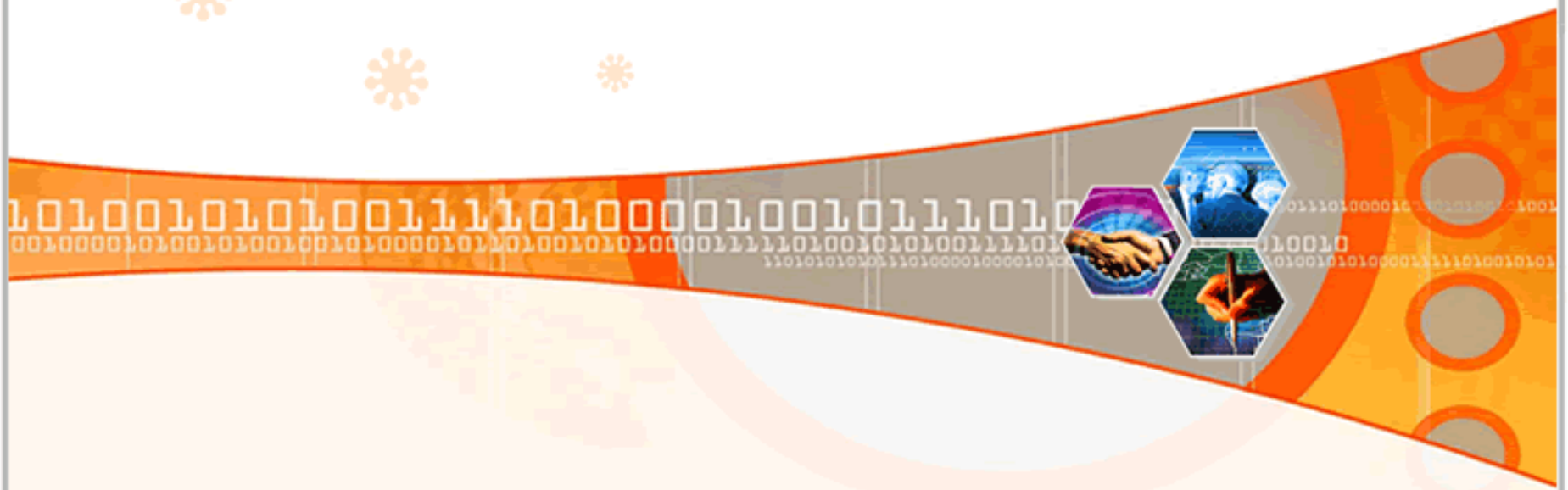
第18章

命令模式



主讲教师：程细柱　韶关学院计算机系
本书主编：刘　伟　　清华大学出版社

本章教学内容

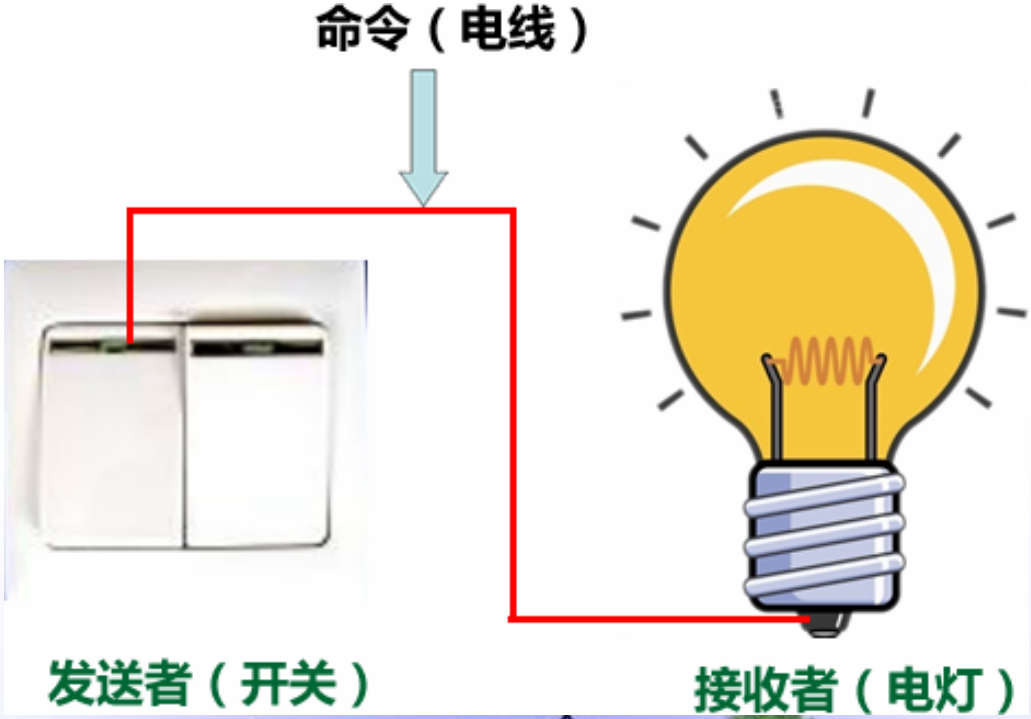
- 命令模式
  - ✓ 模式动机与定义
  - ✓ 模式结构与分析
  - ✓ 模式实例与解析
  - ✓ 模式效果与应用
  - ✓ 模式扩展

命令模式

- 模式动机
  - ✓在软件设计中，我们经常需要向某些对象发送请求，但是并不知道请求的接收者是谁，也不知道被请求的操作是哪个，我们只需在程序运行时指定具体的请求接收者即可，此时，可以使用命令模式来进行设计，使得请求发送者与请求接收者消除彼此之间的耦合，让对象之间的调用关系更加灵活。

命令模式

- 模式动机



命令模式

- 模式动机
  - ✓命令模式可以对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。这就是命令模式的模式动机。

命令模式

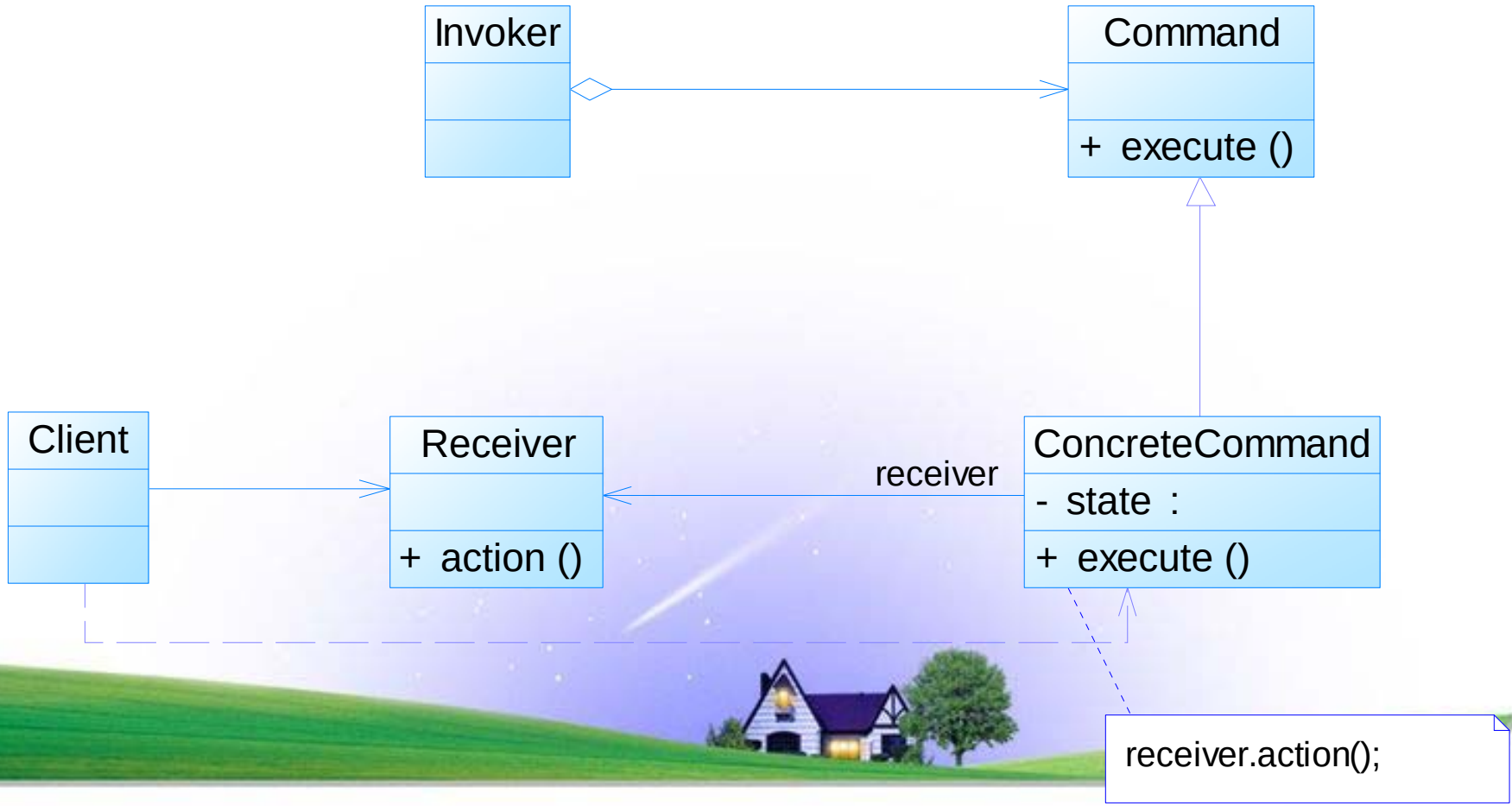
- 模式定义
  - ✓命令模式(Command Pattern)：将一个请求封装为一个对象，从而使我们可用不同的请求对客户进行参数化；对请求排队或者记录请求日志，以及支持可撤销的操作。命令模式是一种对象行为型模式，其别名为动作(Action)模式或事务(Transaction)模式。

命令模式

- 模式定义
  - ✓ **Command Pattern: Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations.**
  - ✓ **Frequency of use: medium high**

命令模式

- 模式结构


Invoker
Command
+ execute ()
Client
Receiver
+ action ()
receiver
ConcreteCommand
- state :
+ execute ()
receiver.action();

命令模式

- 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
    - Client:客户类

命令模式

- 模式分析
  - ✓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。
  - ✓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。
  - ✓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。

命令模式

- 模式分析
  - ✓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。
  - ✓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。

命令模式

- 模式分析
  - ✓典型的抽象命令类代码:

```
public abstract class Command
{
    public abstract void execute();
}
```

命令模式

- 模式分析
  - ✓ 典型的调用者代码:

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

命令模式

- 模式分析
  - ✓ 典型的具体命令类代码:

```
public class ConcreteCommand extends Command
{
    private Receiver receiver;
    public void execute()
    {
        receiver.action();
    }
}
```

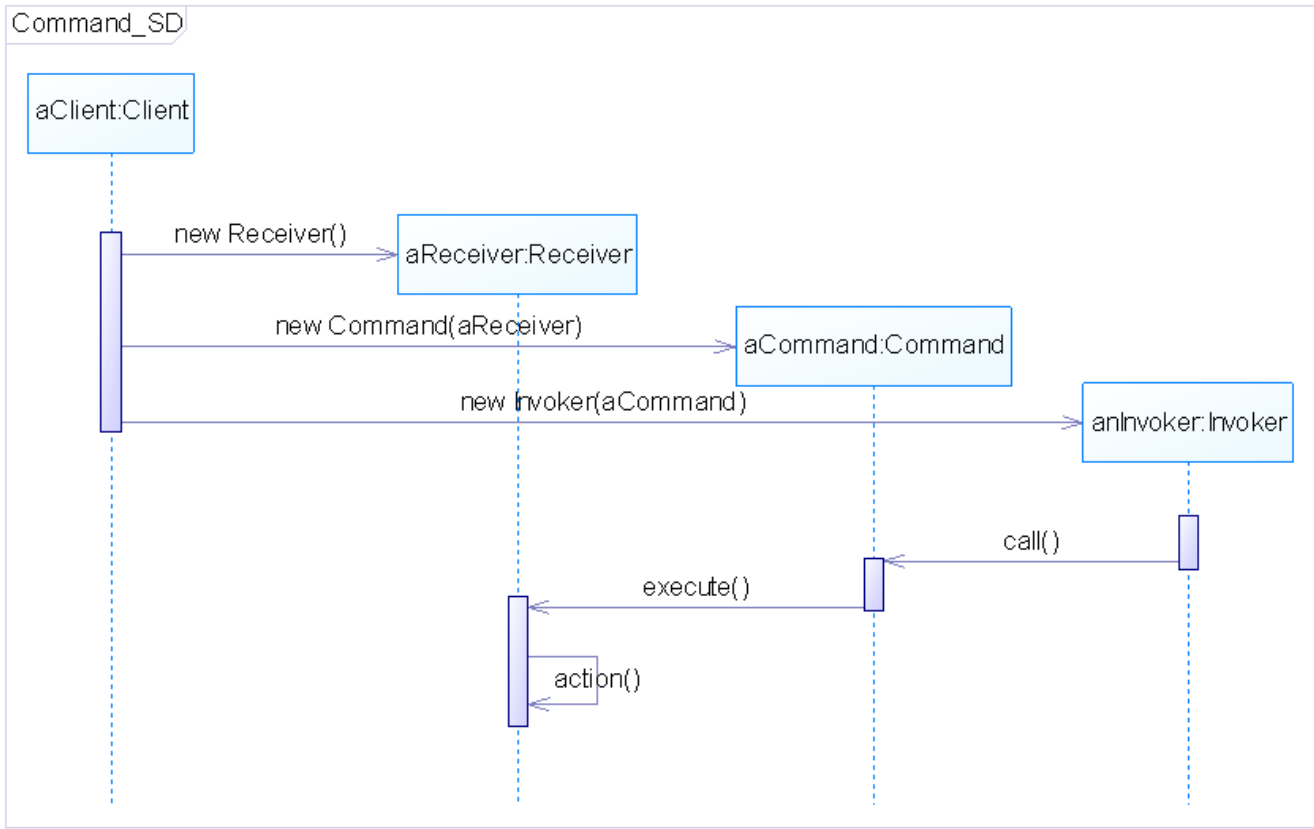
命令模式

- 模式分析
  - ✓ 典型的请求接收者代码:

```
public class Receiver
{
    public void action()
    {
        //具体操作
    }
}
```

命令模式

- 模式分析
  - ✓ 命令模式顺序图 ：



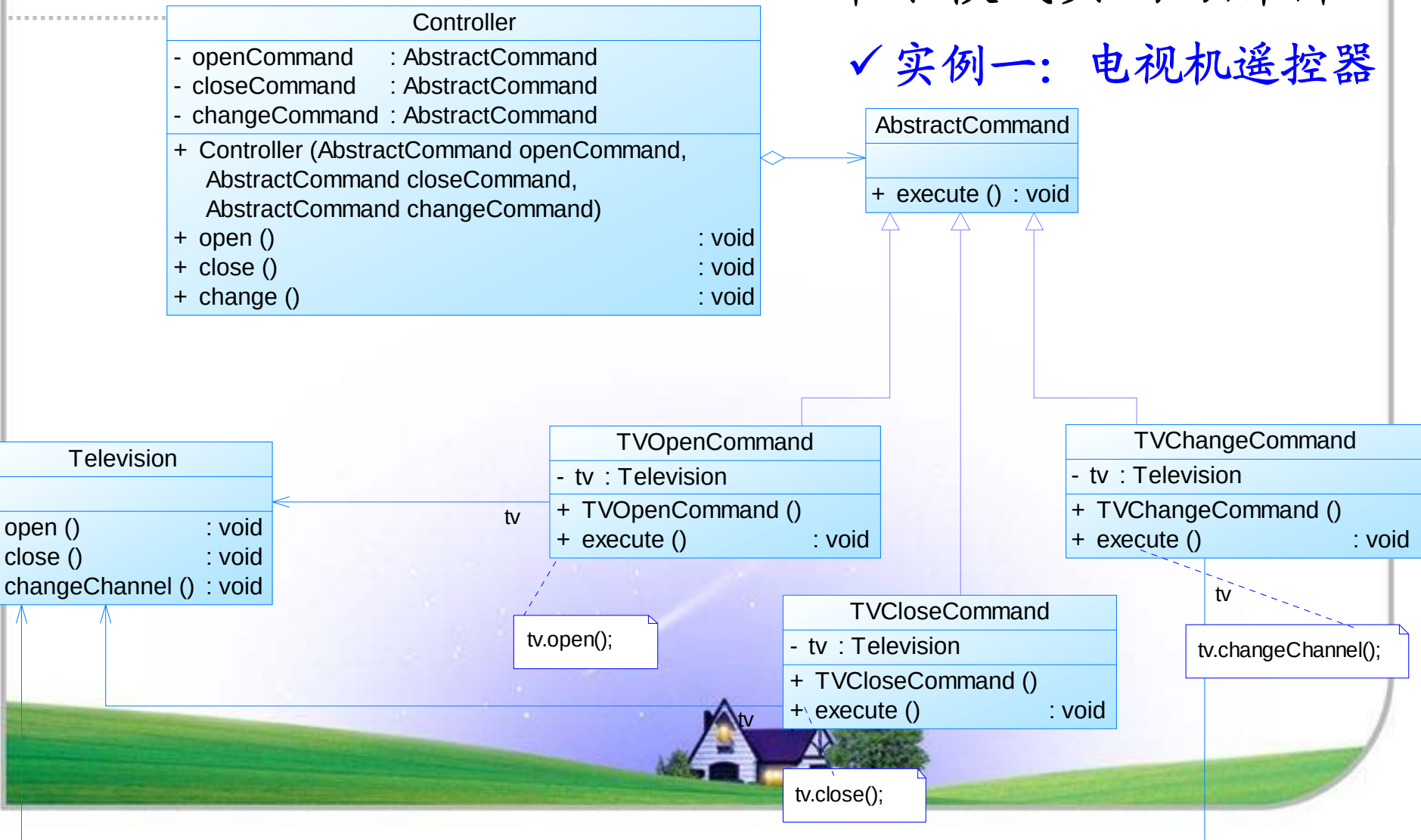
命令模式

- 命令模式实例与解析
  - ✓ 实例一：电视机遥控器
    - 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

◆ 命令模式实例与解析

✓ 实例一：电视机遥控器



命令模式

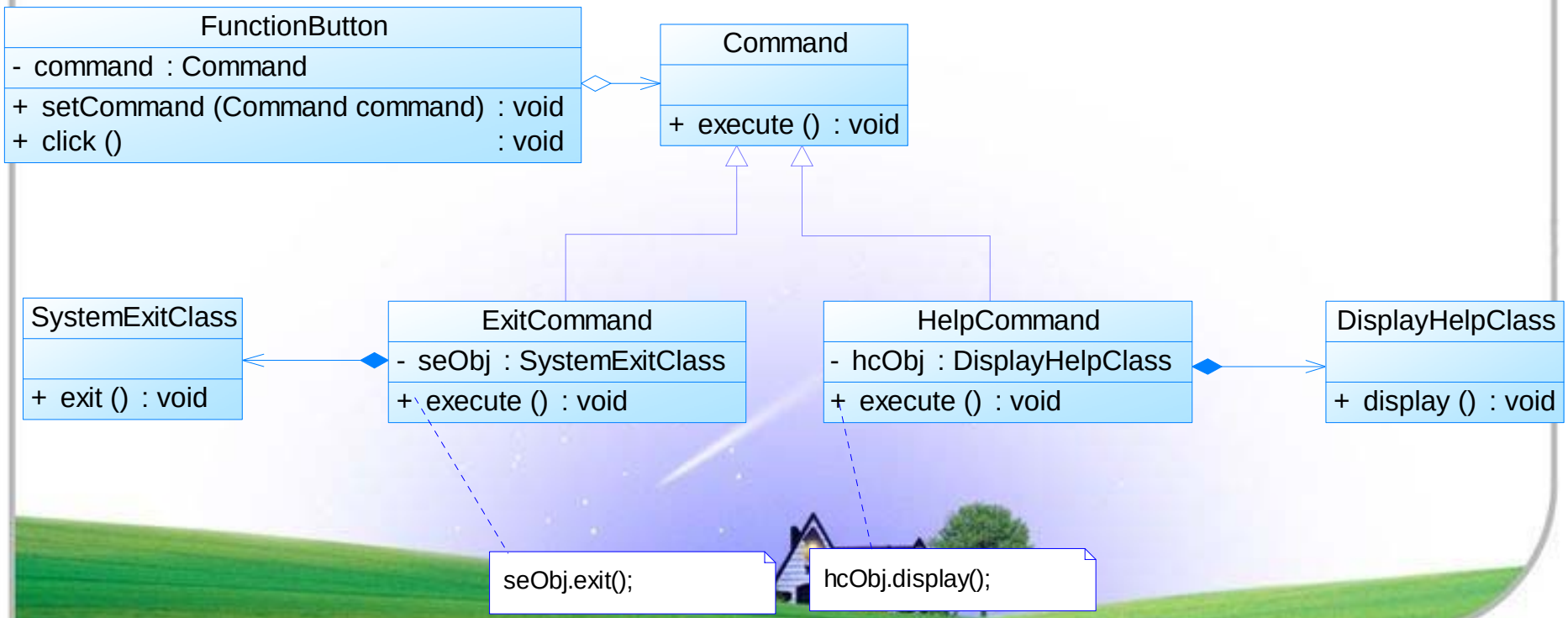
- 命令模式实例与解析
  - ✓ 实例一：电视机遥控器
    - 参考代码：Chapter 18 Command\sample01
    - 下载地址：http://download.csdn.net/user/cflynn

命令模式

- 命令模式实例与解析
  - ✓ 实例二：功能键设置
    - 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，如功能键FunctionButton可以用于退出系统(SystemExitClass)，也可以用于打开帮助界面(DisplayHelpClass)。用户可以通过修改配置文件来改变功能键的用途，现使用命令模式来设计该系统，使得功能键类与功能类之间解耦，相同的功能键可以对应不同的功能。

命令模式

- 命令模式实例与解析
  - ✓ 实例二：功能键设置



命令模式

- 模式优缺点
  - ✓命令模式的优点
    - 降低系统的耦合度。
    - 新的命令可以很容易地加入到系统中。
    - 可以比较容易地设计一个命令队列和宏命令（组合命令）。
    - 可以方便地实现对请求的Undo和Redo 。

命令模式

- 模式优缺点
  - ✓命令模式的缺点
    - 使用命令模式可能会导致某些系统有过多的具体命令类。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。

命令模式

- 模式适用环境
  - ✓在以下情况下可以使用命令模式：
    - 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。
    - 系统需要在不同的时间指定请求、将请求排队和执行请求。
    - 系统需要支持命令的撤销(Undo)操作和恢复(Redo)操作。
    - 系统需要将一组操作组合在一起，即支持宏命令。

命令模式

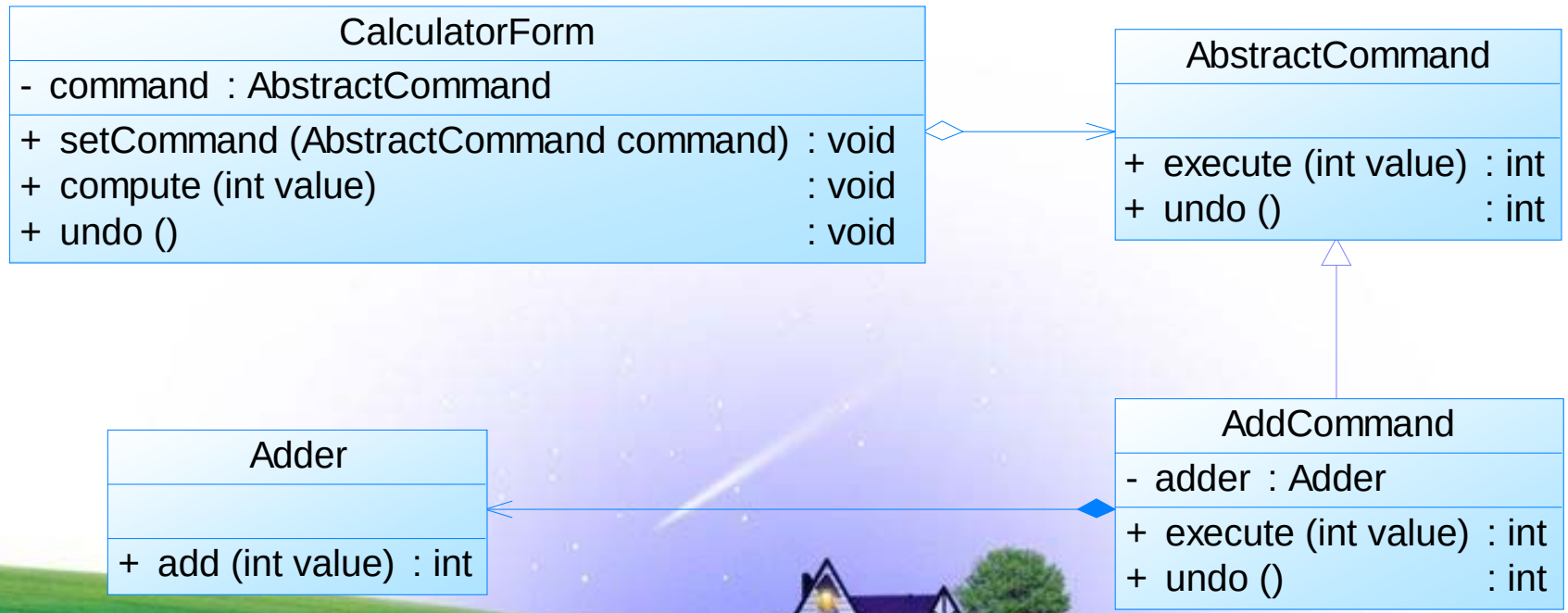
- 模式应用
  - ✓ (1) Java语言使用命令模式实现AWT/Swing GUI的委派事件模型 (Delegation Event Model, DEM) 。
    - 在AWT/Swing中，Frame、Button等界面组件是请求发送者，而AWT提供的事件监听器接口和事件适配器类是抽象命令接口，用户可以自己写抽象命令接口的子类来实现事件处理，即实现具体命令类，而在具体命令类中可以调用业务处理方法来实现该事件的处理。对于界面组件而言，只需要了解命令接口即可，无须关心接口的实现，组件类并不关心实际操作，而操作由用户来实现。

命令模式

- 模式应用
  - ✓(2) 很多系统都提供了宏命令功能，如UNIX平台下的Shell编程，可以将多条命令封装在一个命令对象中，只需要一条简单的命令即可执行一个命令序列，这也是命令模式的应用实例之一。

命令模式

- 模式扩展
  - ✓撤销操作的实现



命令模式

- 模式扩展
  - ✓撤销操作的实现
    - 参考代码：Chapter 18 Command\UndoDemo
    - 下载地址：**http://download.csdn.net/user/cflynn**

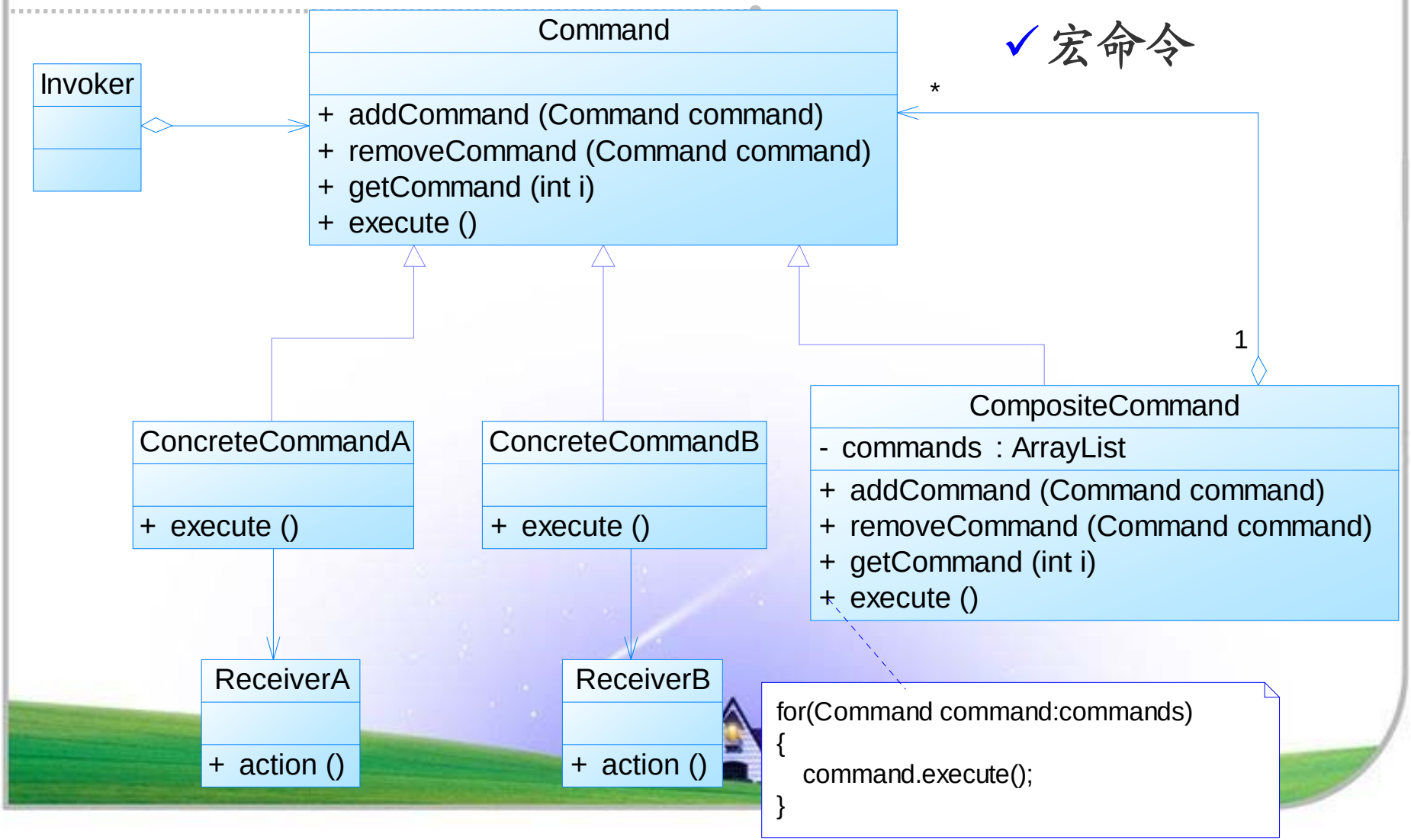
演示.....

命令模式

- 模式扩展
  - ✓ 宏命令又称为组合命令，它是命令模式和组合模式联用的产物。
  - ✓ 宏命令也是一个具体命令，不过它包含了对其他命令对象的引用，在调用宏命令的execute()方法时，将递归调用它所包含的每个成员命令的execute()方法，一个宏命令的成员对象可以是简单命令，还可以继续是宏命令。执行一个宏命令将执行多个具体命令，从而实现对命令的批处理。

命令模式

- 模式扩展
  - ✓宏命令



本章小结

- 在命令模式中，将一个请求封装为一个对象，从而使我们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。命令模式是一种对象行为型模式，其别名为动作模式或事务模式。
- 命令模式包含四个角色：抽象命令类中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；具体命令类是抽象命令类的子类，实现了在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；调用者即请求的发送者，又称为请求者，它通过命令对象来执行请求；接收者执行与请求相关的操作，它具体实现对请求的业务处理。

本章小结

- 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。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。
- 命令模式的主要优点：在于降低系统的耦合度，增加新的命令很方便，而且可以比较容易地设计一个命令队列和宏命令，并方便地实现对请求的撤销和恢复；其主要缺点：在于可能会导致某些系统有过多的具体命令类。
- 命令模式适用情况包括：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操作和恢复操作；需要将一组操作组合在一起，即支持宏命令。

END

Thanks!