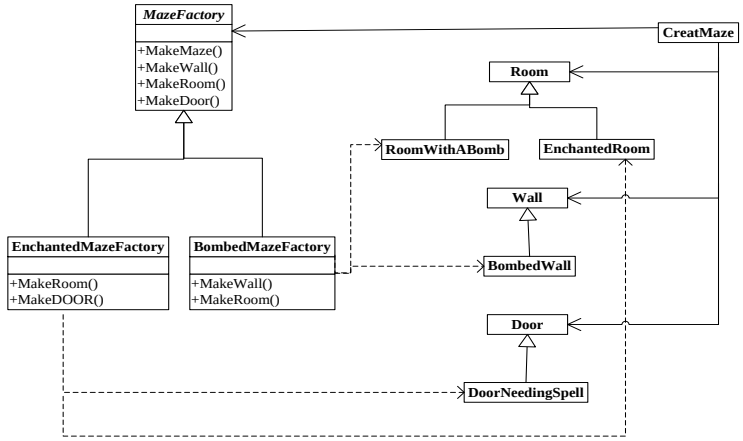
一．Abstract Factory 与 Factory Method 的比较

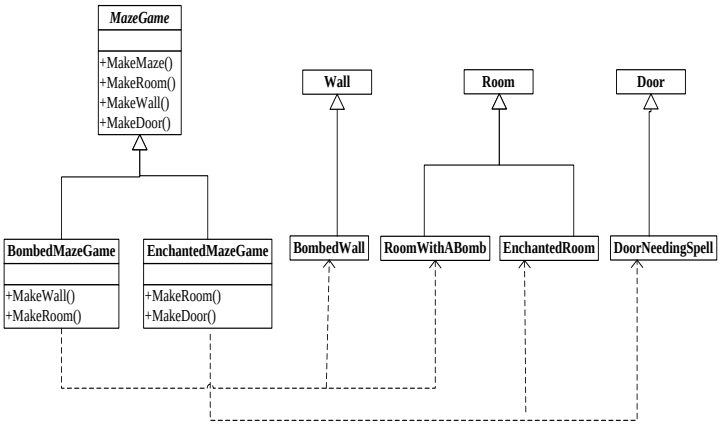
Abstract Factory 的意图：提供创建一系列相关或相互依赖对象的接口，而无需指定它们具体的类。

Factory Method 的意图：定义一个用于创建对象的接口,让子类决定实例化哪一个类.使一个类的实例化延迟到其子类。

比较：1 Abstract Factory 模式可以看作是 Factory Method 模式的一个集合。此模式封装了类构造并将具体类与客户程序通过抽象接口完全分离。**2** Abstract Factory 类通常用 Factory Method 来实现。**3**Abstract Factory 是接口继承，抽象类通常不实现任何功能，仅仅定义一系列接口，子类实现抽象类定义的接口。**4**Factory Method 是实现继承，抽象类实现大部分操作，通常仅将对象的创建工作延迟到子类来完成。

例子：

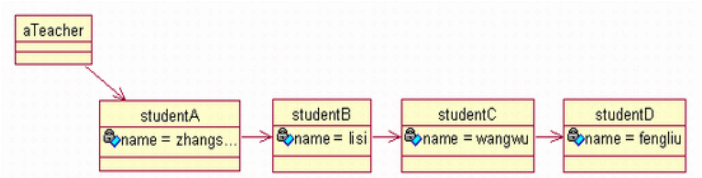




二．Chain of Responsibility（职责链）

职责链模式是一种对象的行为模式。在职责链模式里，很多对象由每一个对象对其下家的引用而连接起来形成一条链。请求在这个链上传递，直到链上的某一个对象决定处理此请求。发出这个请求的客户端并不知道链上的哪一个对象最终处理这个请求，这使得系统可以在不影响客户端的情况下动态地重新组织链和分配责任

例子一：老师把本子传给同学，同学根据本子是不是自己的来进行处理，如果是自己的便留下，否则继续往下传，它解除了老师和同学之间的具体关联关系，老师不需要与本子所属的那个同学直接联系，而是只把本子发给前排的同学，就可以假定已经发给了相应的同学，在同学传递本子的时候，检查本子是不是自己的，如果不是，就传递给下一位。



解决方案：

例子三：机械硬币分拣银行使用职责链。这里并不是为每一种硬币分配一个滑槽，而是仅使用一个滑槽。当硬币落下时，硬币被银行内部的机械导向至适当的接收盒。

例子四：一个小孩子想吃苹果，那他会向家里面的成员请求，家里的成员包括妈妈、爸爸、奶奶、爷爷，这些请求就会沿着这条链传送下去。

三．Flyweight (享元)

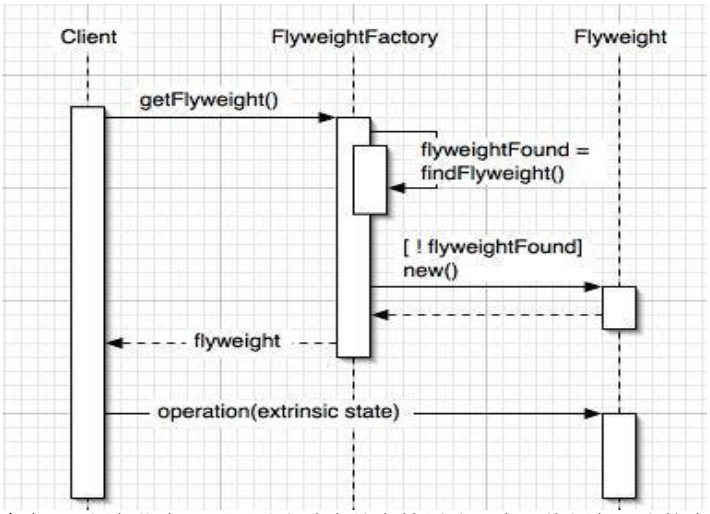
意图：运用共享技术有效的支持大量细粒度的对象。

例子一：Rose 等建模工具中,经常要把图形对象化操作，比如说生成 10000 条直线

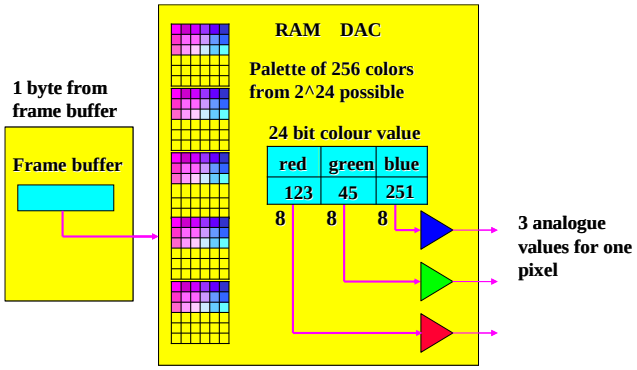
解决方案一：每有需要，都生成一新的直线对象以供调用。

缺点：对每条直线对象都分配内存，有浪费的嫌疑，而且性能低下，不易维护。

解决方案二：现在直线对象可从直线工厂中获得，直线工厂以颜色为共享区，以直线点的位置为非共享区保存直线，这样就提供了极大的灵活性，我们不必每次都创建一个新对象。



例子二：颜色信息可用两种方法存储在帧缓冲器中：将颜色码直接存储在帧缓冲器中。把颜色码放在一个独立的表(调色板)中并用像素值作为这个表的索引。帧缓冲区中的值作为颜色查找表的索引。存储在颜色查找表中的信息控制 RGB 三个电子枪电子的强度。（FlyWeight 的实现原理类同，本身只存储索引，索引指向缓冲区的实际对象）



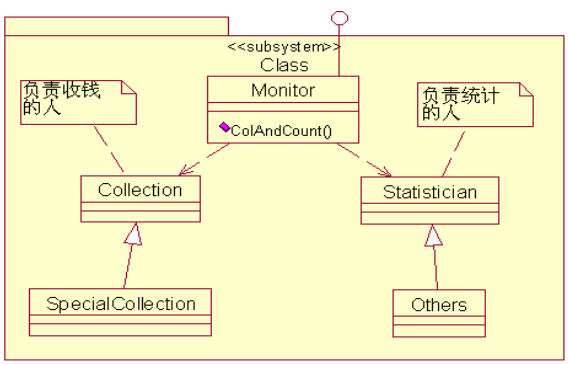
Digital to analogue converters

一一幅 200×200 的 16 色图像，如果每个像素的 RGB 分量都有 256 个级别，则需要的存储空间为：200×200×3=120000Byte。如果采用颜色查找表，则需要的存储空间为：图像的空间 200×200×4/8=20000 Byte，颜色查找表的空间 16×3=48 Byte，共 20048 Byte

四．FAÇADE

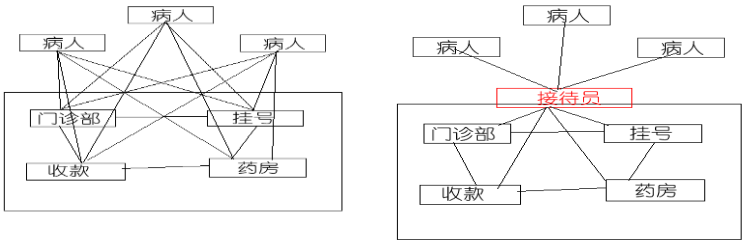
外观模式定义了一个将子系统的一组接口集成在一起的高层接口，以提供一个一致的界面。通过这个界面，其他系统可以方便地调用子系统中的功能，而忽略子系统内部发生的变化。

例子一：学校教务要对每个同学收取某项费用，要求每个人都要限期交到教务，具体怎么实施？**解决方案：**每个同学以个人为单位，自行到学校教务交纳。以班级为单位，先由班长将本班费用收集，统计好姓名，然后统一交到教务



Collection 和 Statistician 只是充当概括类，并非抽象类。它们不做具体的工作，只负责与 Monitor 进行沟通，由它们派生来的子类才进行具体工作。

例子二：1.如果把医院作为一个子系统，按照部门职能，这个系统可以划分为挂号、门诊、划价、化验、收费、取药等。看病的病人要与这些部门打交道，就如同一个子系统的客户端与一个子系统的各个类打交道一样，不是一件容易的事情。2.首先病人必须先挂号，然后门诊。如果医生要求化验，病人必须首先划价，然后缴款，才能到化验部门做化验。化验后，再回到门诊室。3.解决这种不便的方法便是引进外观模式。可以设置一个接待员的位置，由接待员负责代为挂号、划价、缴费、取药等。这个接待员就是外观模式的体现，病人只接触接待员，由接待员负责与医院的各个部门打交道。



例子三：

比如我们日常生活中经常见的例子：

消费者使用电话订货，拨打一个号码与客服代表联系，客服代表则扮演了这个“外观”，他包含了与订货部、收银部和送货部的接口。

五．Decorator(装饰)

装饰模式:Decorator 常被翻译成"装饰"，翻译成"油漆工"更形象点,油漆工(decorator)是用来刷油漆的,那么被刷油漆的对象我们称 decoratee,这两种实体在 Decorator 模式中是必须的。动态给一个对象添加一些额外的职责,就象在墙上刷油漆.使用 Decorator 模式相比用生成子类方式达到功能的扩充显得更为灵活。正如 Decorator（装饰）模式的名字暗示的那样，Decorator 模式可以在我们需要为对象添加一些附加的功能/特性时发挥作用，除此之外，更为关键的是 Decorator 模式研究的是如何以对客户透明的方式动态地给一个对象附加上更多的特性，换言之，客户端并不会觉得对象在装饰前和装饰后有什么不同。Decorator 模式可以在不创造更多子类的情况下，将对象的功能加以扩展。Decorator 模式使用原来被装饰的类的一个子类的实例，把客户端的调用委派到被装饰类，Decorator 模式的关键在于这种扩展是完全透明的。正因为 Decorator 模式可以动态扩展 decoratee 所具有的特性，有人将其称为“动态继承模式”，该模式基于继承，但与静态继承下进行功能扩展不同，这种扩展可以被动态赋予 decoratee。

例子一：最近家里搞装修，做了一套家具，需要刷一下漆，因此我就去市场找了油漆师傅和油漆徒弟两个人。油漆师傅主要买油漆和调油漆，油漆徒弟主要来刷油漆。

A：油漆徒弟。刷油漆的工作主要是由油漆徒弟来完成，所以我们把油漆徒弟定义成 Brusher.

B：油漆师傅。我们把油漆师傅定义成 Decorator。

例子二：咖啡店需要售卖各种各样的咖啡：黑咖啡、加糖、加冰、加奶、加巧克力等等。顾客要买咖啡，他可以往咖啡任意的一种或几种产品。

SugarDecorator sugar(new IceDecorator(new Coffee()));

SugarDecorator sugar(new SugauDecorator(new IceDecorator(new Coffee())));

aSugar->bSugar->aIce->aCoffee

六．Composite

意图：将对象以树形结构组织起来,以达成“部分－整体” 的层次结构，使得客户端对单个对象和组合对象的使用具有一致性。

例子一：老师让学生去打扫卫生。对老师来说，让一个学生去打扫卫生和让一个组的学生去打扫卫生是一样的， 他可不用去到小组里一个一个的去通知学生们。

例子二：component：考研的参考书。leaf:《考研数学复习指南》《考研数学题型集粹》。composite：考研数学题型集粹与练习题集、考研英语词汇阅读全攻略、考研三科复习指南……诸如此类的东东，就是把几本书打包在一起。

七．Singleton

意图：保证一个类仅有一个实例，并提供一个该实例的全局访问点。

动机：在软件系统中，经常有这样一些特殊的类，必须保证它们在系统中只存在一个实例，才能确保它们的逻辑正确性、以及良好的效率。（这应该是类设计者的责任，而不是使用者的责任。） 为了达到这个目的就引入了单件。

适用性：当类只能有一个实例而且客户可以从一个众所周知的访问点访问它时。 当这个唯一实例应该是通过子类化可扩展的，并且客户应该无需更改代码就能使用一个扩展的实例时。

实现：保证有唯一的实例。创建实例的操作隐藏在一个类操作后面，由它保证只有一个实例被创建。这个操作访问保存唯一的实例的变量，而且它可以保证这个变量在返回值之前用这个唯一实例初始化。

代码举例：

SINGLETON 类的子类实现扩展

a. 使用单件注册表 Singleton 类可以根据名字在一个注册表中注册他们单件实例，这个注册表在字符串名字和单件之间建立映射。

b. 采用动态链接的方法

例子一：利用单件模式，这个类里的方法和属性在内存都只有一个对象，所以单件模式的应用场景是对于应用系统中某个对象是唯一的或是固定的，比如数据库的连接，应用程序的配置文件，WEB 的计数器功能等之类的应用。

采用静态成员函数而不是静态对象实现的原因：静态对象不能保证实例的唯一性。可能没有足够的信息在静态初始化时实例化每个单件。单件之间没有依赖关系（没有定义转换单元上全局对象的构造器调用顺序）。所有单件无论用到与否都要被创建。

建一个 WEB 页，在其 PAGE_LOAD 方法里写上：

运行后，会发现 TextBox 里显示的值是修改后的"This is a test!",可见前者对该对象变量做改变之后，后来调用者则取到的是修改后的最新值，说明在内存中只有一个对象实例存在。

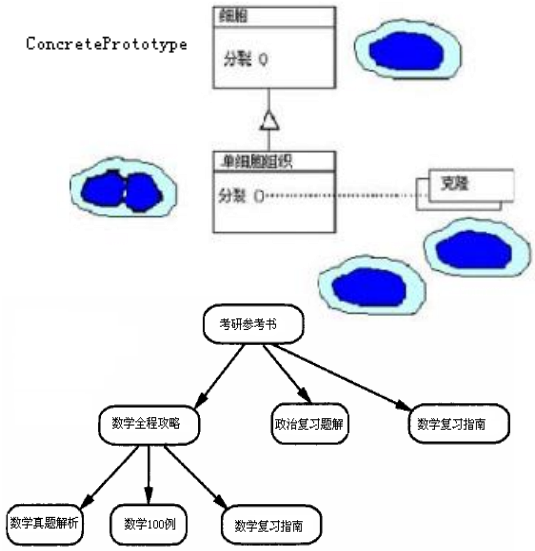
例子二：美国总统的职位是 Sigleton，美国宪法规定了总统的选举，任期以及继任的顺序。这样，在任何时刻只能由一个现任的总统。无论现任总统的身份为何,其头衔"美利坚合众国总统"是访问这个职位的人的一个全局的访问点。

八. Prototype 用原型实例指定创建对象的种类,并且通过拷贝这些原型创建新的对象。Prototype 模式允许一个对象再创建另外一个可定制的对象，根本无需知道任何如何创建的细节。工作原理是：通过将一个原型对象传给那个要发动创建的对象，这个要发动创建的对象通过请求原型对象拷贝它们自己来实施创建。

例子一：假设有一家店铺是配钥匙的,他对外提供配制钥匙的服务(提供 Clone 接口函数),你需要配什么钥匙它不知道,他只是提供这种服务,具体需要配什么钥匙只有到了真正看到钥匙的原型才能配好.也就是说,需要一个提供这个服务的对象,同时还需要一个原型(Prototype),不然不知道该配什么样的钥匙。

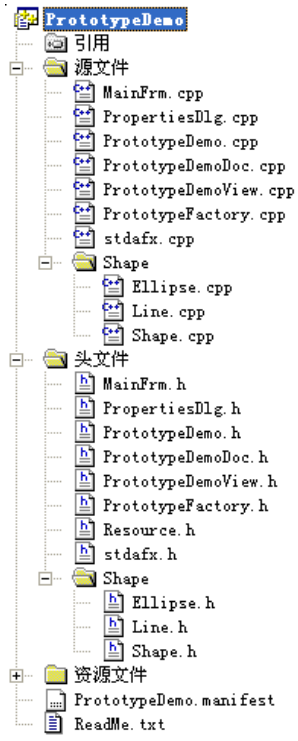
例子二：正如一个细胞的有丝分裂，产生两个同样的细胞，是一个扮演主动角色复制自己原型的例子，这演示了原型模式。一个细胞分裂，产生两个同样基因型的细胞。换句话说，细胞克隆了自己。

例子三：假定开发一个调色板，用户单击调色板上任一个方块，将会返回一个对应的颜色的实例。



九、图形系统

涉及模式：Prototype (原形模式)、AbstractFactory (抽象工厂模式)。**目录结构：**解开压缩包以后，看到如下结构：



其中 Shape.h ， Line.h ， Ellipse.h 三个头文件是声明了图形类的，我们只实现了画直线和圆，不过已经很有代表性了，要加其它的也很容易，只要再新建类就行了，由于和调用没有任何关系，框架类只需要知道抽象类 CShape 就行了，至于有几个子类它不管，再加上又工厂 CPrototypeFactory 类负责构造，所以要添加新图形类，只需要从 CShape 派生就行了。

其它信息请参加源程序，有详细注释和说明。

功能说明:

主要功能都在“功能”这一菜单项上，功能有：生成直线、生成圆，删除图形对象。

按下生成直线以后，程序视图区出现了一条直线。点击此直线对象，则它会被选中，可以自由拉伸和移动。还可以复制粘贴，支持快捷键。

在“文件”菜单项中还能保存图形文件和打开图形文件。

程序流程图：详细设计 算法函数 编码前最后一部分

业务流程图：eventflow 需求阶段 某个用例业务的基本流程。概要阶段

系统流程图：当前系统的状态，在可研报告阶段，当前系统可能是手工系统、计算机系统等。不主张

系统流程图（System Flowchart）：是描绘系统物理模型的传统工具。它的基本思想是用图形符号以黑盒子形式描绘系统里面的每个部件(程序、文件、数据库、表格、人工过程等)，表达信息在各个部件之间流动的情况和系统的操作控制。

系统流程图：是描述系统物理模型的一种传统工具。它是表达数据在系统各部件之间流动的情况，而不是对数据加工处理的控制过程，它是物理数据流图而不是程序流程图。系统流程图形象的呈现了软件的功能，即使不懂软件的人也可以轻松的看懂，可以说它是软件设计师与用户之间沟通、交流的有效工具。

开关与电灯、排气扇示意图

分析：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 ←请求发送者。电灯 ←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。

软件开发：按钮 ←请求发送者。事件处理类←请求的最终接收和处理者。

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。

动机：1.将请求发送者和接收者完全解耦 2.发送者与接收者之间没有直接引用关系 3.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。

命令模式为动作(Action)模式或事务(Transaction)模式、“用不同的请求对客户进行参数化”、“对请求排队”、“记录请求日志”、“支持可撤销操作”。

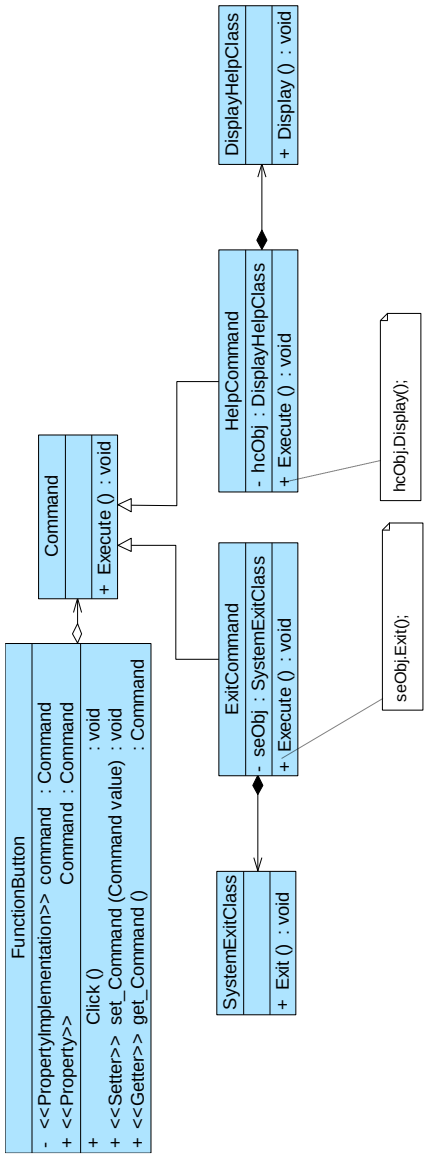
结构：1.命令模式的本质是对请求进行封装 2.一个请求对应于一个命令，将发出命令的责任和执行命令的责任分开 3.命令模式允许请求的一方和接收的一方独立开来，使得请求的一方不必知道接收请求的一方的接口，更不必知道请求如何被接收、操作是否被执行、何时被执行，以及是怎么被执行的。

命令模式的实现

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 类来实现），也可以用于显示帮助文档（由 DisplayHelpClass 类来实现）。

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，可为同一个功能键设置不同的功能。右上角图。



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。（下页 1）

动机：

1.当一个请求发送者发送一个请求时，有不止一个请求接收者产生响应，这些请求接收者将逐个执行业务方法，完成对请求的处理 2.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 3.批处理

实现命令队列

记录请求日志：动机：

将请求的历史记录保存下来，通常以日志文件(Log File)的形式永久存储在计算机中。1.为系统提供一种恢复机制 2.可以用于实现批处理 3.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

实现：1.将发送请求的命令对象通过序列化写到日志文件中 2.命令类必须使用属性[Serializable]标记为可序列化。命令对象—序列化→日志文件

实现撤销操作：实例：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。

设计一个简易计算器，该计算器可以实现简单的数学运算，还可以对运算实施撤销操作。

宏命令：动机：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。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，还可以继续是宏命令。

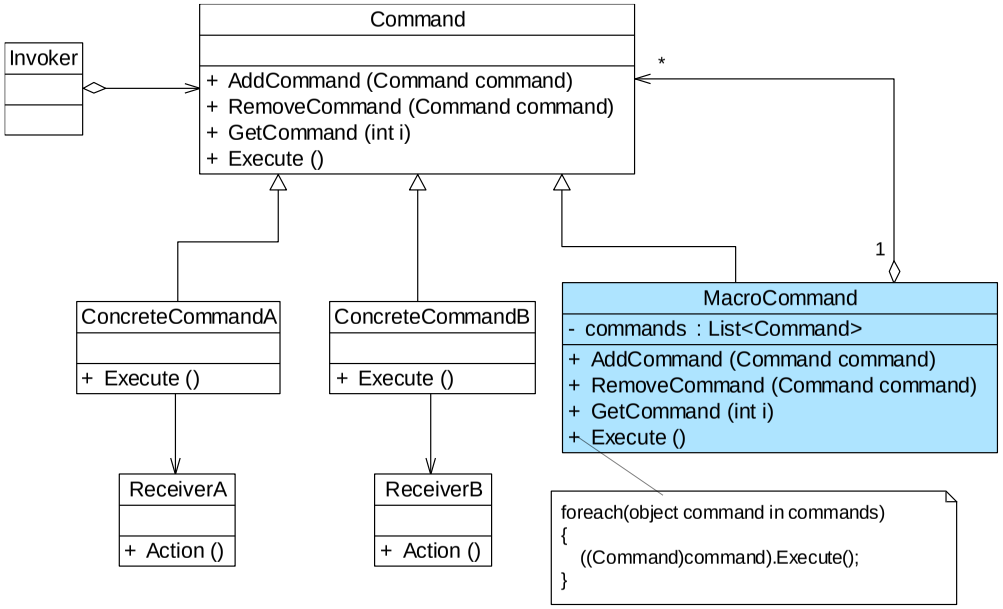
执行一个宏命令将触发多个具体命令的执行，从而实现对命令的批处理

宏命令。**结构：**

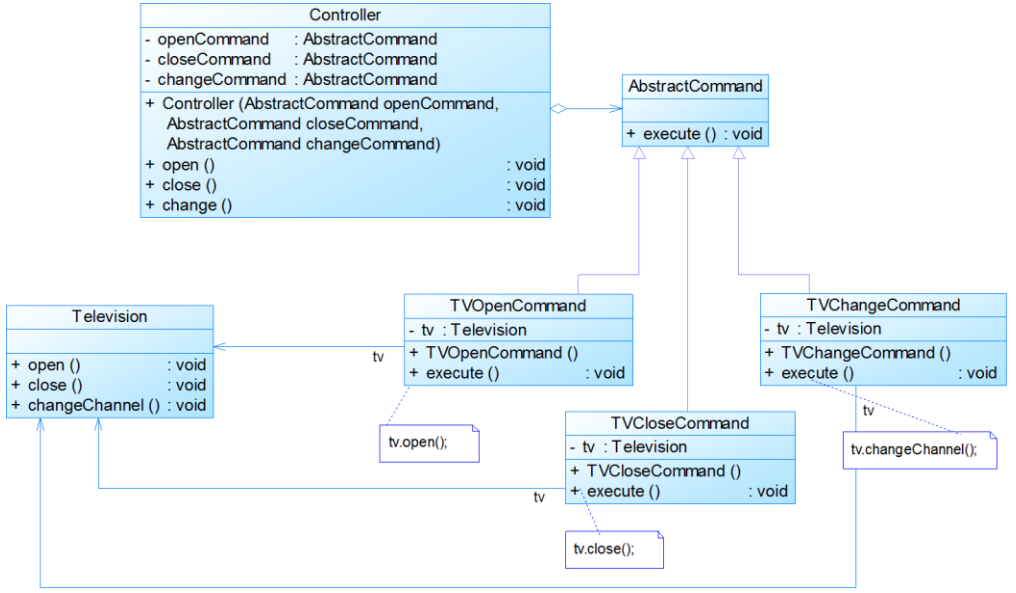
模式优点：1.降低了系统的耦合度 2.新的命令可以很容易地加入到系统中，符合开闭原则 3.可以比较容易地设计一个命令队列或宏命令（组合命令）4.为请求的撤销(Undo)和恢复(Redo)操作提供了一种设计和实现方案

模式缺点：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（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）2.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 3.系统需要在不同的时间指定请求、将请求排队和执行请求 4.系统需要支持命令的撤销(Undo)操作和恢复(Redo)操作 5.系统需要将一组操作组合在一起形成宏命令

宏命令图



电视机图



简易计算器图