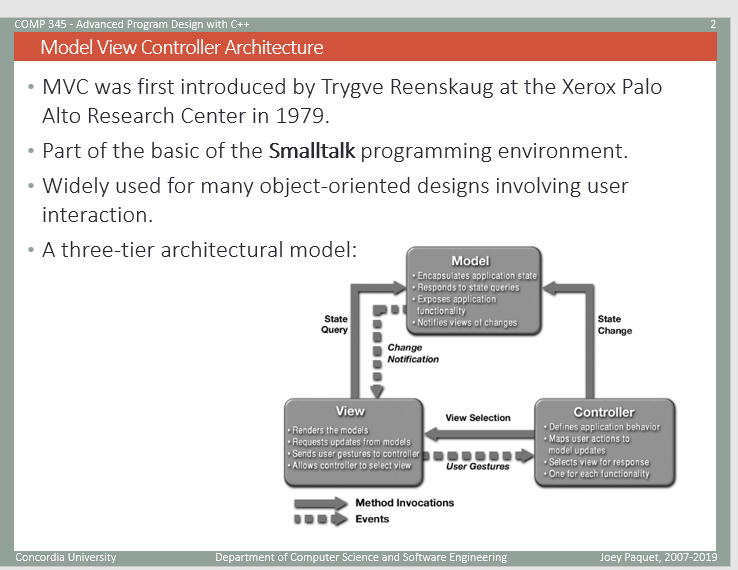
MVC是smalltalk programming environment的基础

多用于object-oriented designs包括用户交互

三层结构



Model: 模型

管理application domain//应用领域 的behavior与data （behavior就是实际的function，data就是ID啊啥的比较具体的内容（也是到现在为止一直写的内容））

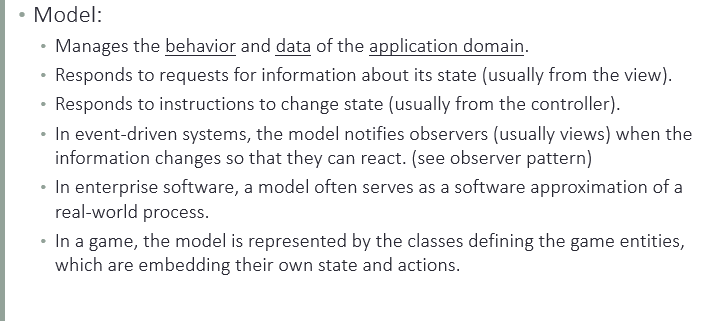
回复那些查询他当前state信息的请求（通常都是客户视角）

回复那些改变state的指令（通常来自controller）

在event-driven的系统中，model 会告知observer 什么时候信息改变，这样observer就可以对改变做出回应（see observer pattern）

在企业级的系统中，model用来充当对真实世界的模拟

在游戏中，model是用class描述的游戏实体



notify 告知

View:

把model render成一个可视化或交互的形式，通常是一个user interface element(例如网页页面)

一个单一model可以存在许多不同目地 的view

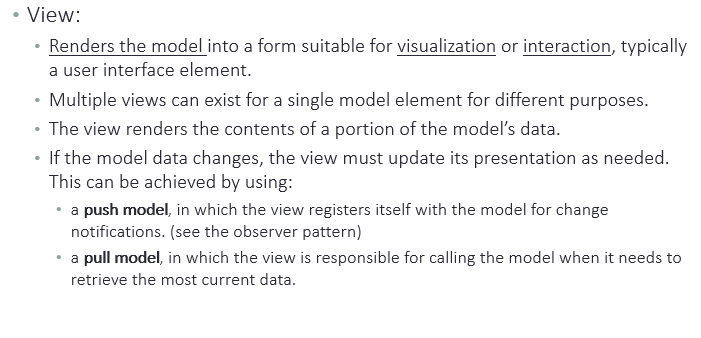
view 会把model里面隐藏的数据用可视的方式表现出来

如果model data改变了，view也会相对改变他展示出来的数据

Update可以通过

Push model: view 自己观察model的改变并做出相应改变

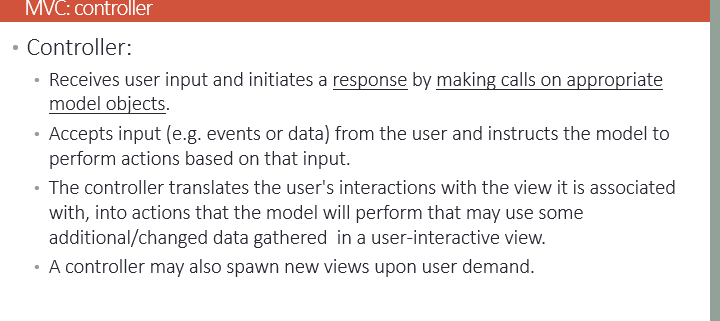
Pull model： view负责手动的去call model当他需要最新数据的时候



portion部分

Initiate a response发起一个响应

Spawn 生成



Controller可以得到user input并且做出相对响应——通过call 对应的model objects

接受input(event or data)并且指导model做出基于Input的操作

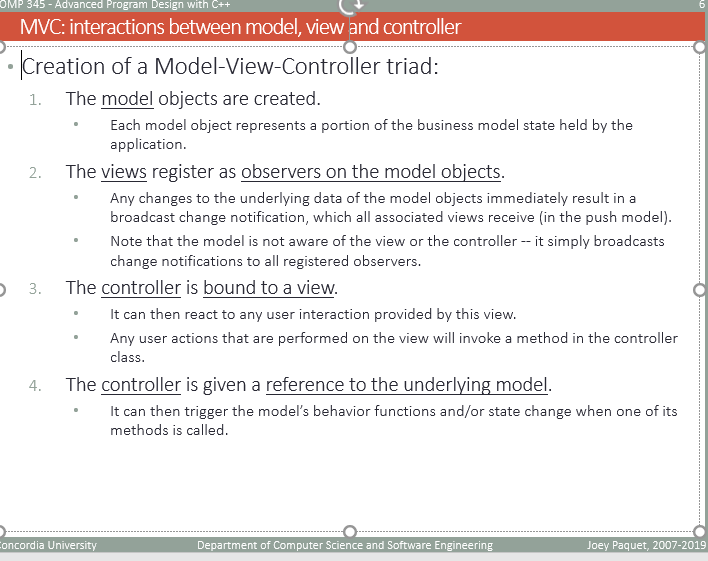
Controller把用户对view的操作翻译成model将要进行的动作，这些动作或许会从用户对view的操作中收集信息

Controller也可以根据用户需要生成新的view

Underlying潜在的

Broadcast 散播，广播，大范围的

Notification 通知



先创造model object

每一个model object代表着应用的business model state的一部分

View作为对model object 登记的observer（只有登记过后的observer才能被检测到）

任何对model object潜在数据的改动都会瞬间导致大范围的改动通知，所有有关的view都会收到（在push model里）//pull model只有view主动申请才会得到改动

注意model本身并不能意识到view或controller的存在——他只是单纯的大范围的给所有登记过的 observer改动通知

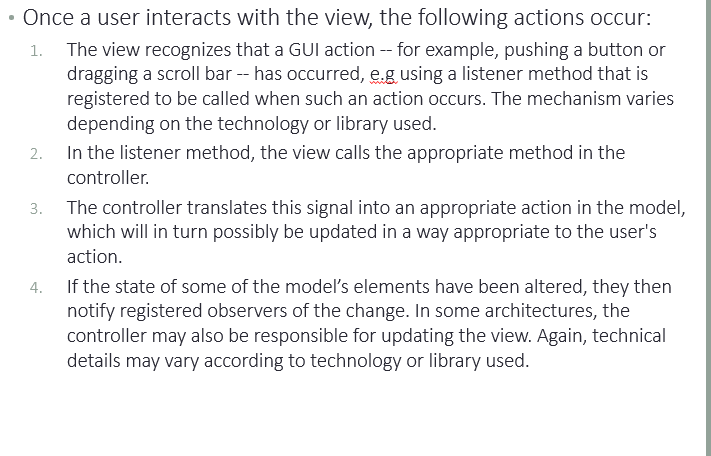
Controller通常与一个view所绑定

Controller会检查测到用户与view的操作，然后做出相对的回应

任何对view进行的user action实际上都是调用controller class的method

Controller 给了一个接触潜在的underlying model的接口

我们可以通过controller的其中一个method来激活model里的function，这样用户在view上的操作就通过controller传递到了model上



当user与view所交互，接下来的行为就会发生：

1.View意识到了一个GUI 动作——例如按一个键或者拉滚动条。当这些行为发生的时候，一个登记过的listener method就会被call

2.在Listener method中，view会call合适的controller method

3.controller把信号转化成model中合适的动作，这样就会让model做出对user动作的更新

4.如果model 中的element被改变了。那么就会告诉registered observers这些改变。在一些结构中，controller或许也负责updating view，再说一遍细节可能与tech或library 有关

总结：model object是无法直接控制的实体

View 是登记在model object上的observer ，有Listener method来接受GUI的操作

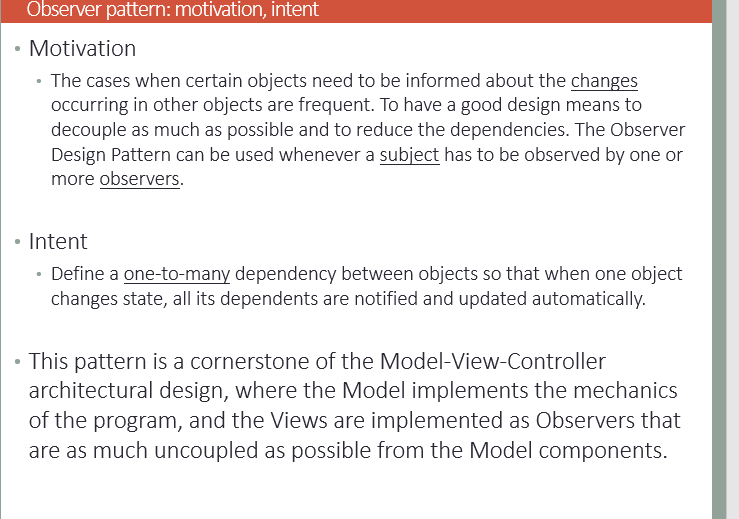
（view就是可视化的窗口啥的，并且能检测GUI操作）

Controller里的method会被listener method所激活，对model object间接控制

Pattern 模式  
motivation 动机

Intent 目的

Cornerstone基石



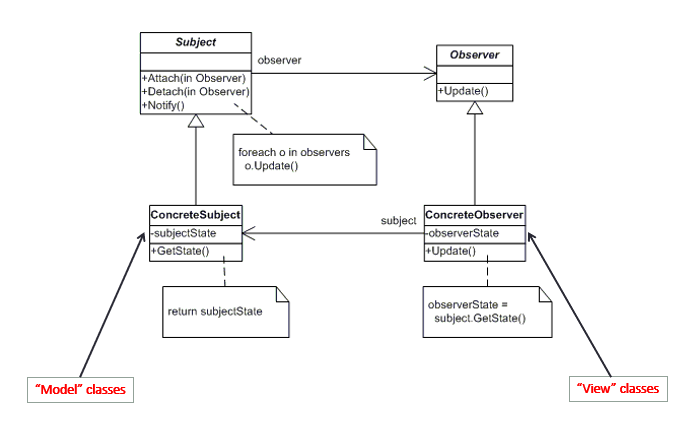
Motivation：动机，我们为什么要observer pattern

有的时候其他object有变化时，要通知别的object （model通知所有view或view通知controller）。这个情况过于频繁了， 而一个良好的设计意味着我们需要减少dependencies。 Observer design depattern 可以使用当 subject必须被一个或多个observer所观测的时候

Intent: 目的

描述一个 one-to-many的依赖关系，当一个object改变状态的时候，其他dependents(依赖这个object的objects们)都会被自动通知并update

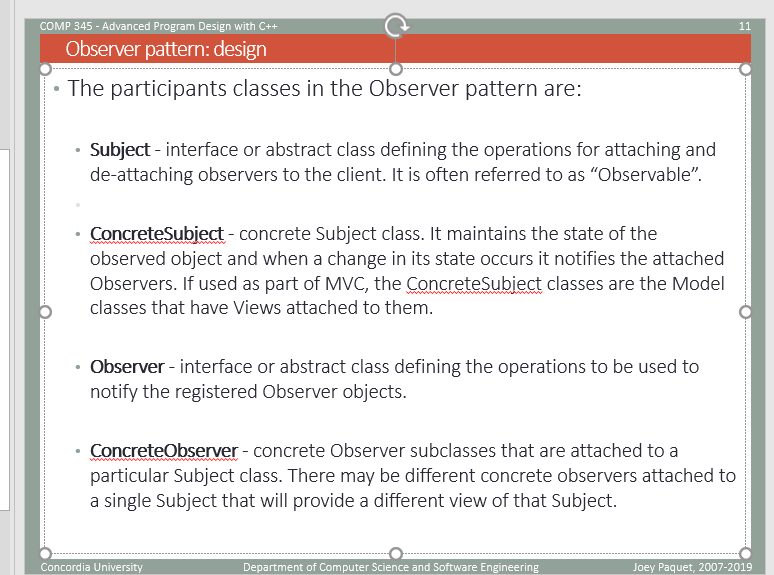
这个模式是model-view-controller的基石。Model描述了这个program的结构，view作为observer并且尽力与model部分分开



Subject 与Observer都是比较原始的抽象父类

Subject是主语的意思，用来attach 或断开observer

Concrete 具体



Observer pattern 由以下几种class构成

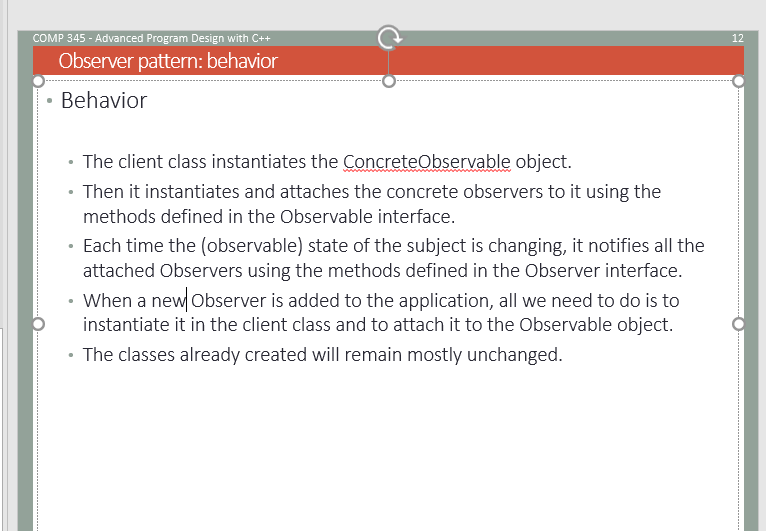
Subject: 抽象类，包含了attach 与de attach observer,

concreteSubject: 具体的 Subject class: 它包含了observed object的 state，当它的state发生变化时，他就会通知所有附着的observer 。 如果在MVC里， ConcreteSubject Class就是那部分有view的Model Class

Observer: 接口或抽象类，描述了怎样提醒那些registered Observer objects 的操作

ConcreteObserver: 具体的对concreteSubject的observer（VIEW），一个subject可能有不同的observer

Instantiate 举例说明 ，初始化



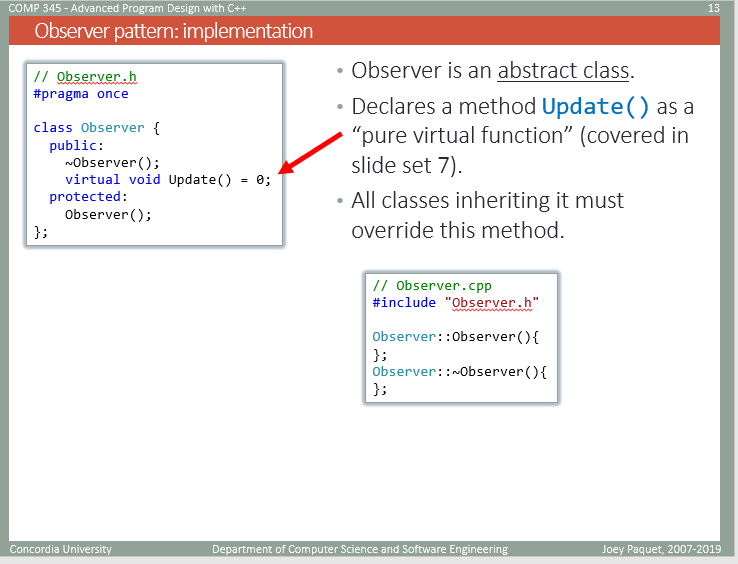
客户端会 初始化一个 concreteObservable object //一个model

然后他会初始化一个observer(GUI浏览器) 并 把observer 连接到这个object（model object）上 使用 observable interface （model class）所描述的Method

每一次subject状态改变，，告诉每个observer,通过observer里的method

当一个新observer加入到应用时，我们只是在client class里初始化他然后连接到observable object他

已经创造过的class大多无法改变

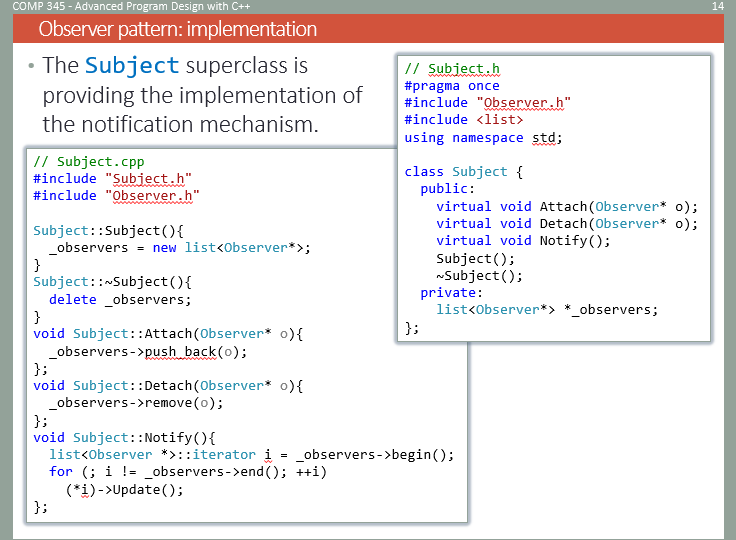


Observer是一个抽象类 (.h)

Update是一个virtual function

所有继承他的class必须override这个virtual method

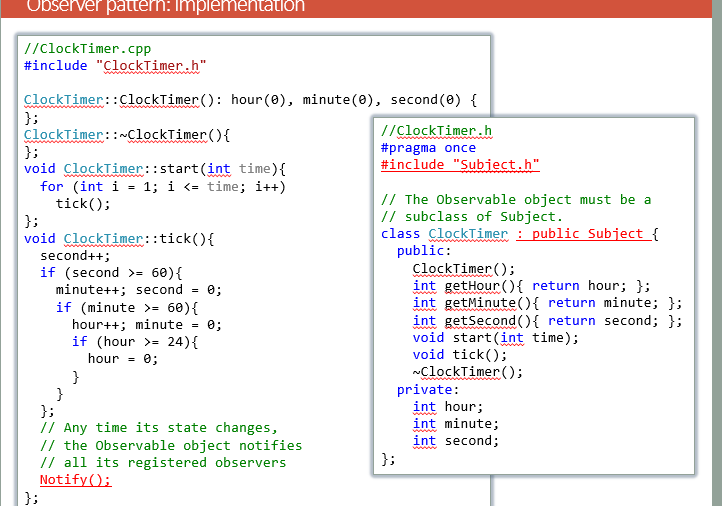
Update(让GUI刷新当前信息)

Subject superclass 用来提供记录有几个observer, //基础抽象父类

然后可以连接，移动observer

也可以提醒observer

（让每一个observer更新当前信息）

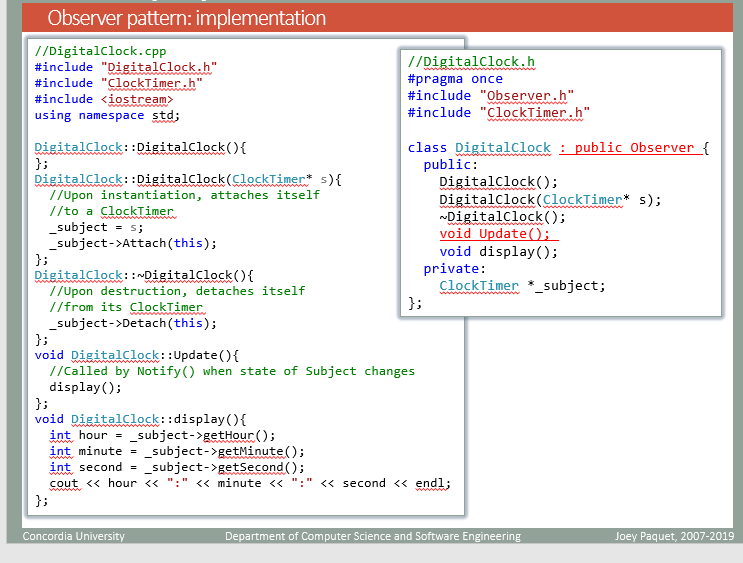


最基本的东西 observable object

当有改动的时候，notify()

Observable object需要是Subject class的子类

我们在描写cpp的时候别的都一样，但如果状态改动的话，就要在最后一步使用notify()，这样才能让每一个SUBJECT里的OBSERVER都刷新

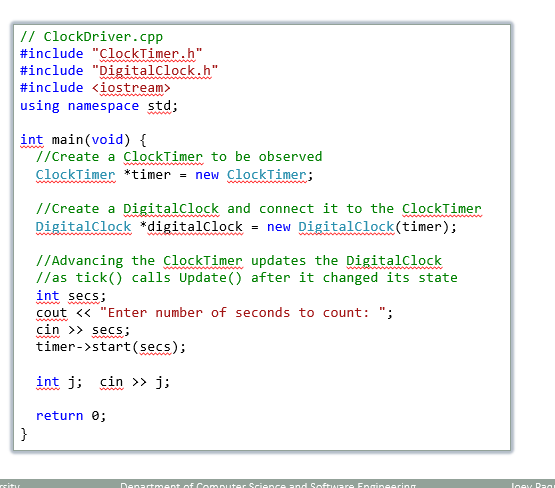


Digital clock是observer的子类

创造它的时候要把他attach到一个subject(实际上这个subject就是具体的clocktimer object)上

Update就是display，每一次notify，就要展示当前信息

Display不应该被主动call，他只是to string



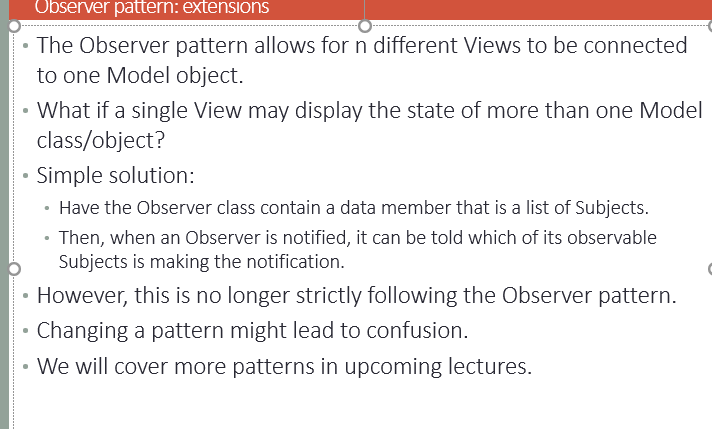
Driver由clocktimer于digitalClock构成

ClockTimer是observable object

DigitalClock是view

先生成背后的model

再生成用来观测的observer并绑定



Observer pattern允许n个不同的view连接到一个model object上

那么一个view像展示多个model的状态呢

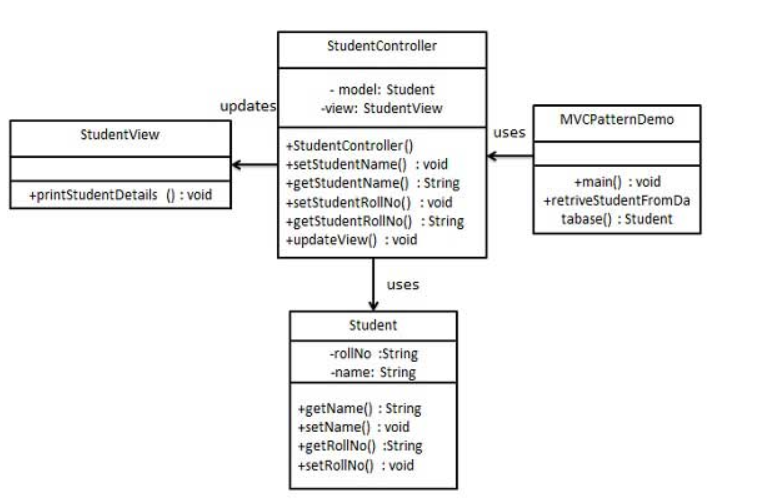
解决方案：

让observer class可以包含subject的list

然后当observer被统治的时候，他可以被告知哪个subject做出了这个通知

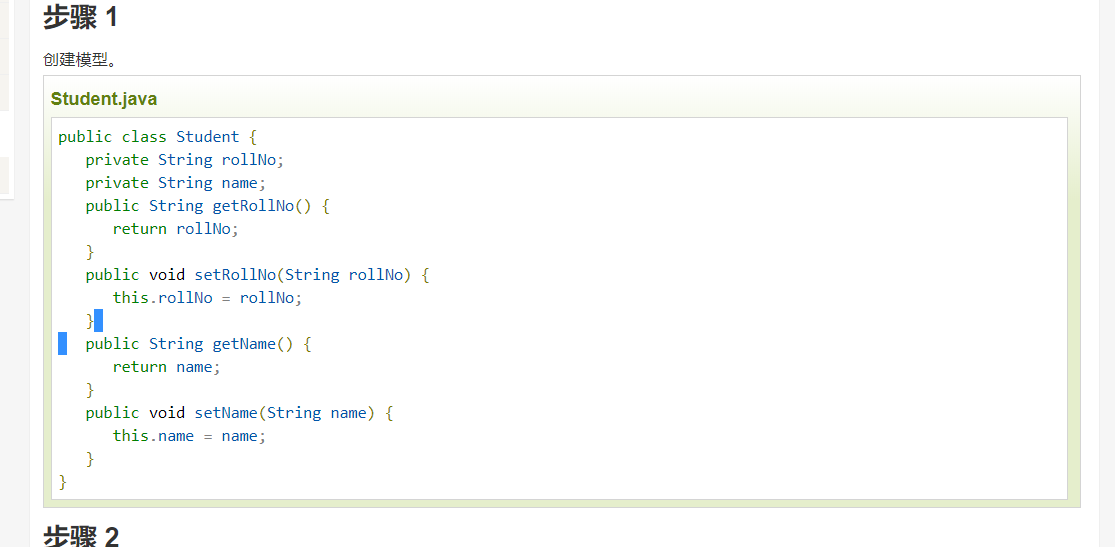
然而这样就不再是标准意义上的observer pattern了

有可能让人困惑



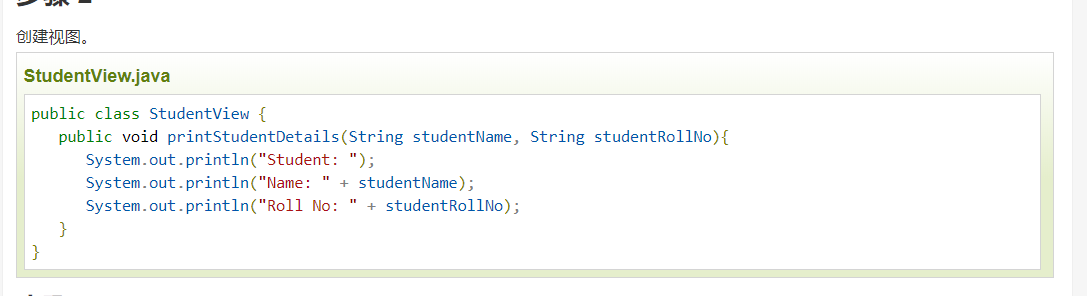
额外理解

下面这个student就是model



‘

VIEW就是输出类



Controller

