案例编号：

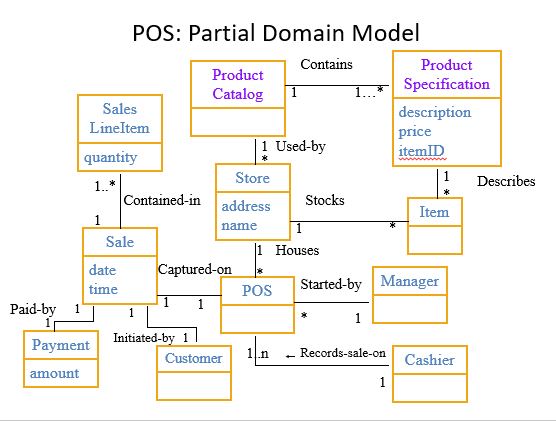
案例名称：销售终端系统协作协议的设计

案例目标：

1. 实现用例/任务/系统事件
2. 确保所需的质量（时间性能，鲁棒性等）

案例描述：

在本案例中主要是针对销售终端系统进行三种模式（专家、创作者、控制）的协议设计，在超市购物的时候，我们会先选择货物，排队交款，然后打出明细单，最后交款，取走货物。其中明细单中列出了所有相关信息：商店、商品名称、规格、数量、单价、终端号、销售人等。商品通常放在仓库里管理，仓库使用POS机进行管理销售，每个POS机都需要有人去启用。销售终端系统（POS：Point of Sale），涉及商店、产品目录、产品规格、商品、出纳员、销售、销售单、付款、顾客。POS机有相应的管理员就行管理，管理员就是专家，可以提供相应的信息查询。订单是由客户发起的，订单可以进行支付。把相应的职责分配给相应的类，销售终端系统的类图如下所示：



Keyword：

POS、销售、商品、协议

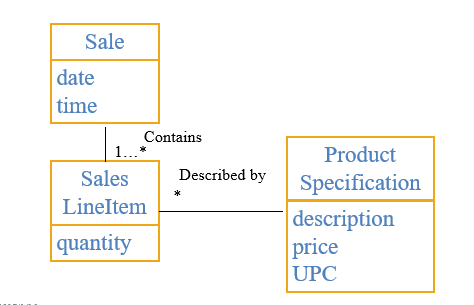
系统描述：

案例建模过程：

1. 先找出系统的类，然后找出涉及销售总和的类。
2. 然后对类的职责进行分配，主要有三种模式（专家、创作者、控制
3. 根据系统的要求用相应的模式进行协议的设计，需要和题目中类的操作和属性一致。
4. 改进设计让系统具有更好的低耦合和高内聚。
5. 最后依据协议进行合同的设计。

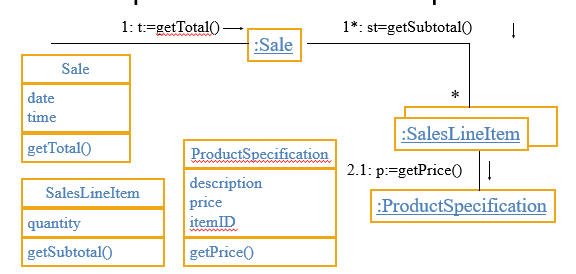
案例结果：

类图：



注：UPC: unique product code

（1）专家协议 – 示例



合同（专家模式）：

Sale::getTotal()

**require**

none

**do**

iterate all the owned SalesLineItems

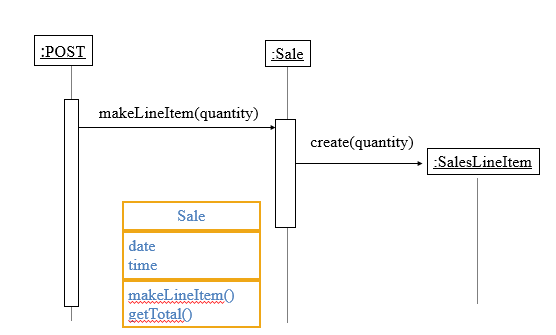
**ensure**

return = sum(slineItem[0].getSubTotal(),…,

slineItem[n-1].getSubTotal())

**end**

（2）创作者协议 – 示例



合同（创作者模式）:

Sale::makeLineItem(quantity:Integer)

**require**

quantity is an integer and larger than 0

**do**

create a new object newItem of SalesLineItem

**ensure**

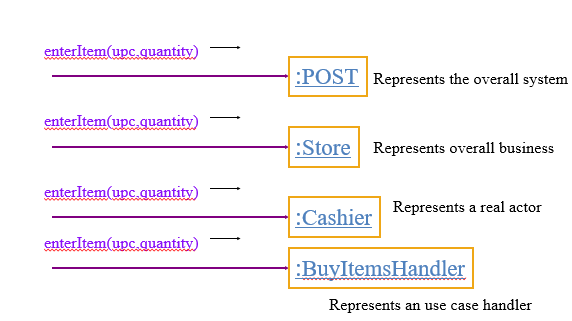
newItem.quantity = quantity;

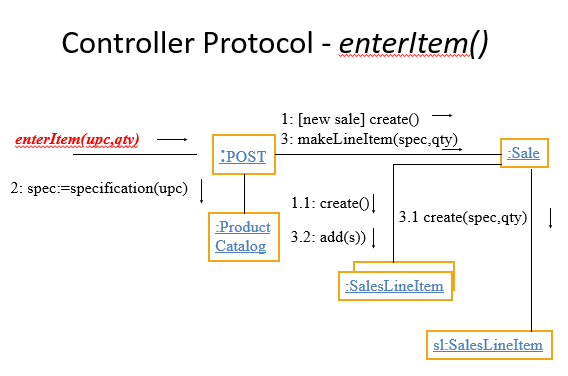
Current.include (newItem);

itemCount = itemCount\_old + 1

**end**

（3）控制协议 – 示例





合同（控制模式）:

POST::enterItem(upc:UPC,qty:Integer)

**require**

upc <> null;

qty is an integer and larger than 0

**do**

query the product spec with upc;

If (new Sale) then create sale object;

record the items into the Sale object

**ensure**

spec.code = upc;

sale.makeLineItem(spec,qty) invoked;

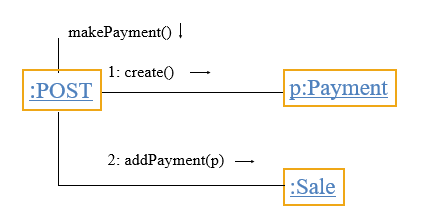
new\_sale = false

**end**

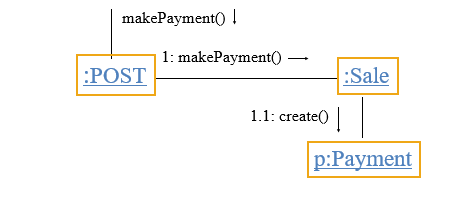
低耦合示例：

创建付款的类：Payment、Post、Sale

第一种解决方案：



第二种解决方案：



高内聚示例：

创建付款可以分配到POST，因为它在真实世界中记录付款。这是与创作者协议相关的指南建议的。这是可以接受的。然而，如果这个逻辑应用于几个其他类似的情况，越来越多的工作可能被分配给一个类。这可能导致一个内聚类。例如，如果有50个系统操作，并且POST做一些与每个操作相关的工作，那么POST将是一个大的内聚类。相比之下，第二种设计，让销售创建付款，支持更高的内聚力在POST。这种设计支持高内聚和低耦合，因此优于第一种。

1. 反面

案例总结：

本案例主要是针对POS机中销售总和进行设计的协议，要想知道销售总和就需要获取总计所需的信息以及所有销售项目的知识。因为销售是信息专家，我们将此责任分配给Sale，SalesLineItem，ProductSpecification可以确定小计。所以确定了案例结果中的三个类（Sale，SalesLineItem，ProductSpecification）。

接下来进行协议的设计，设计分为三种模式。第一种是专家协议，可以从案例结果中看到专家协议的设计，专家协议有助于保留信息的封装，可是得到低耦合的设计。但是在某些情况下，专家协议可能无法提供一个理想的解决方案。在我们的案例中我们需要明确是谁在数据库中保存销售，由于所有保存的信息是在Sale类，它应该是负责保存的信息。这意味着Sale类必须知道处理数据库。但是它违反了“关注分离”原则。实际上，数据库存储在设计上应用由独立的组件来处理，而不是逻辑上的类。还有另外一个问题就是专家做的事情越来越多，设计阶段新被识别的属性也将被加入到专家类中，导致专家承担更多的职责。为了处理actor的请求，系统需要根据类的设计实例化对象。所以我们就引出了创作者协议。创作者协议实际上是专门的专家，即用于分配创建对象的责任。最后是控制协议，控制主要是处理系统事件的非用户界面对象。对于哪个对象应该负责处理enterItem（）系统事件消息这个问题，我们在案例结果中给出了控制协议的示例。在这个模式中我们应该选择控制器来处理系统事件，不要将太多的责任分配给控制器。我们设计的目标是低耦合高内聚，可以更好的实现重用。

对于合同的设计来说，可以通过协议获得合同，需要注意的是前置条件和后置条件是断言，即，它们是可以被评估为真或假的表达式，其中前提条件表达任何调用必须满足的要求，后置条件表示在过程执行结束时确保的属性。如果没有前提条件或后置条件，则前提条件或后条件假定为真（这等于说没有前条件或后条件），如果前提条件或后置条件失败（即，计算结果为假）。