System Analysis and Design

L18.Use Case Realization with GRASP

后期按排

① 6月14日停课

② 6月30日结束课

③ 考核方式：项目报告 + Homework + 期终闭卷 (30+30+40)

④ 项目报告需提交一个最终版(各种文档)

⑤ Homework提交将发布在网盘上.

} 7月5日前提交

⑥ 第3次作业做为期中考查，每人一定要提交，否则登记有问题

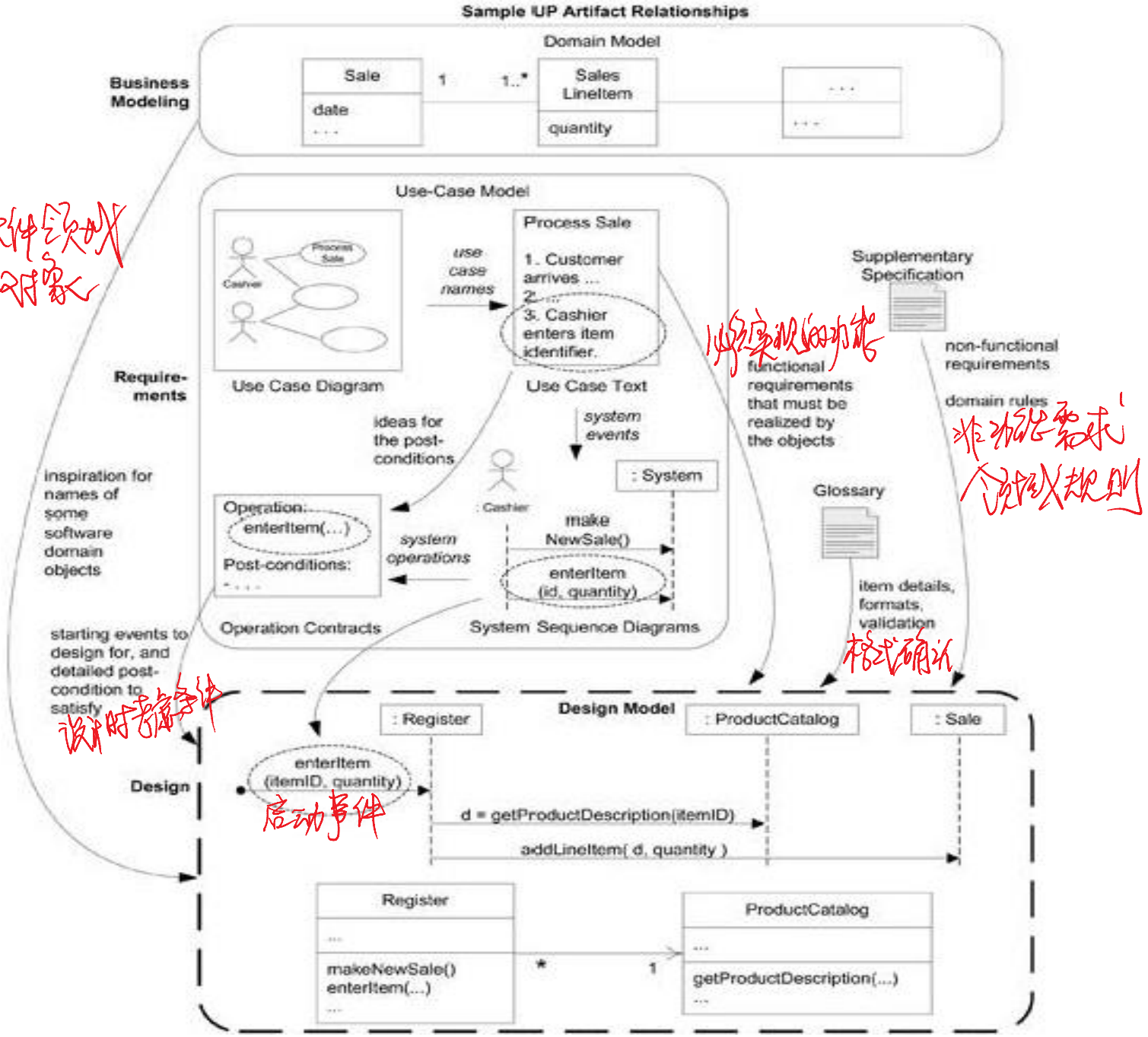
Topics

- Use Case Realization
- Use case realization with GRASP
- Iterative and Evolutionary Object Design

Use Case Realization

- Describes how a particular use case is realized within the design model in terms of collaborating objects 什么是 Usecase Realization
- UML diagrams are a common language to illustrate use case realizations. 表示方法：UML图
- We can apply principles and patterns during this use case realization design work. 使用原理与模式进行设计(用例实现)

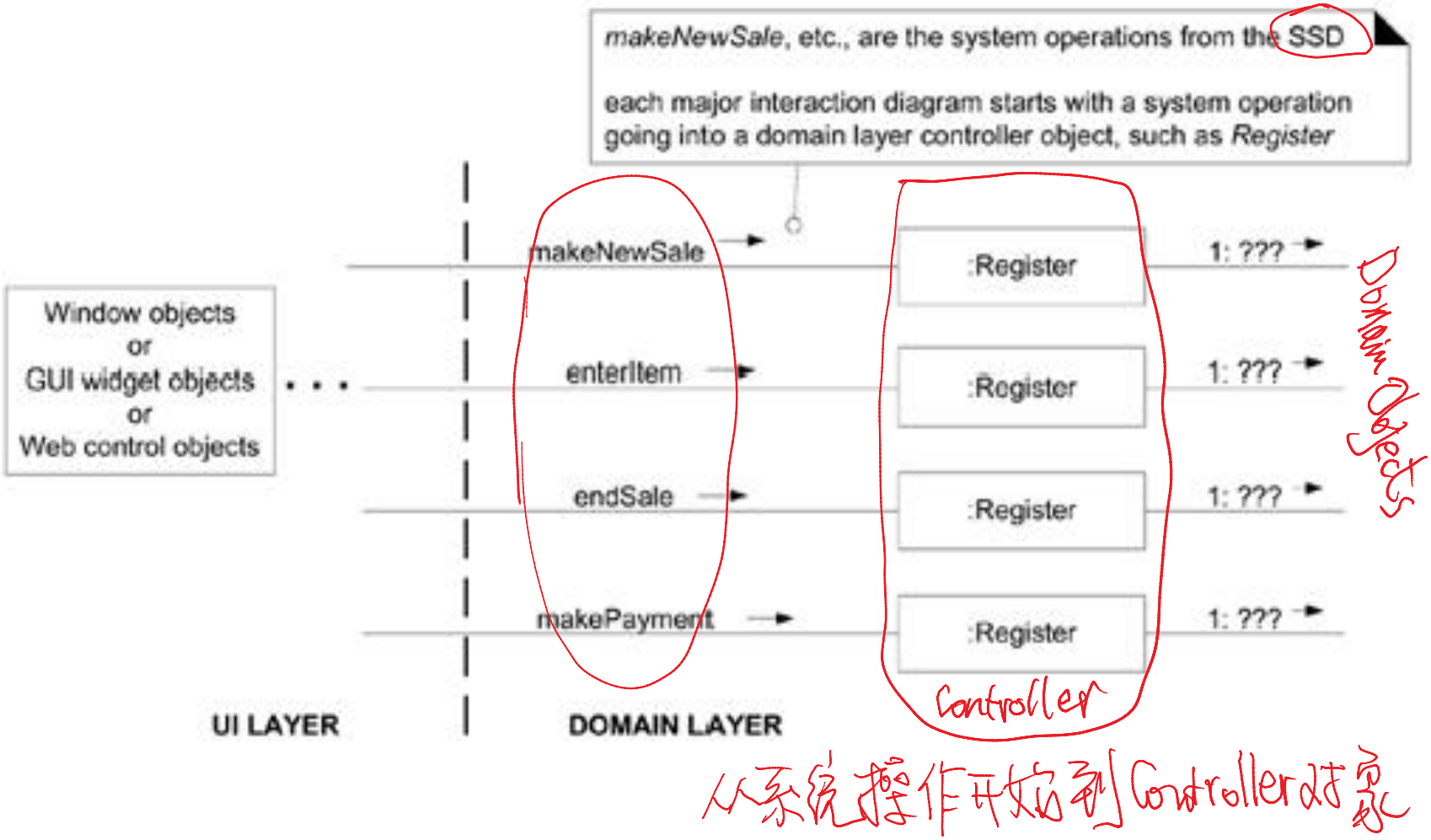
Artifact relationships, emphasizing use case realization



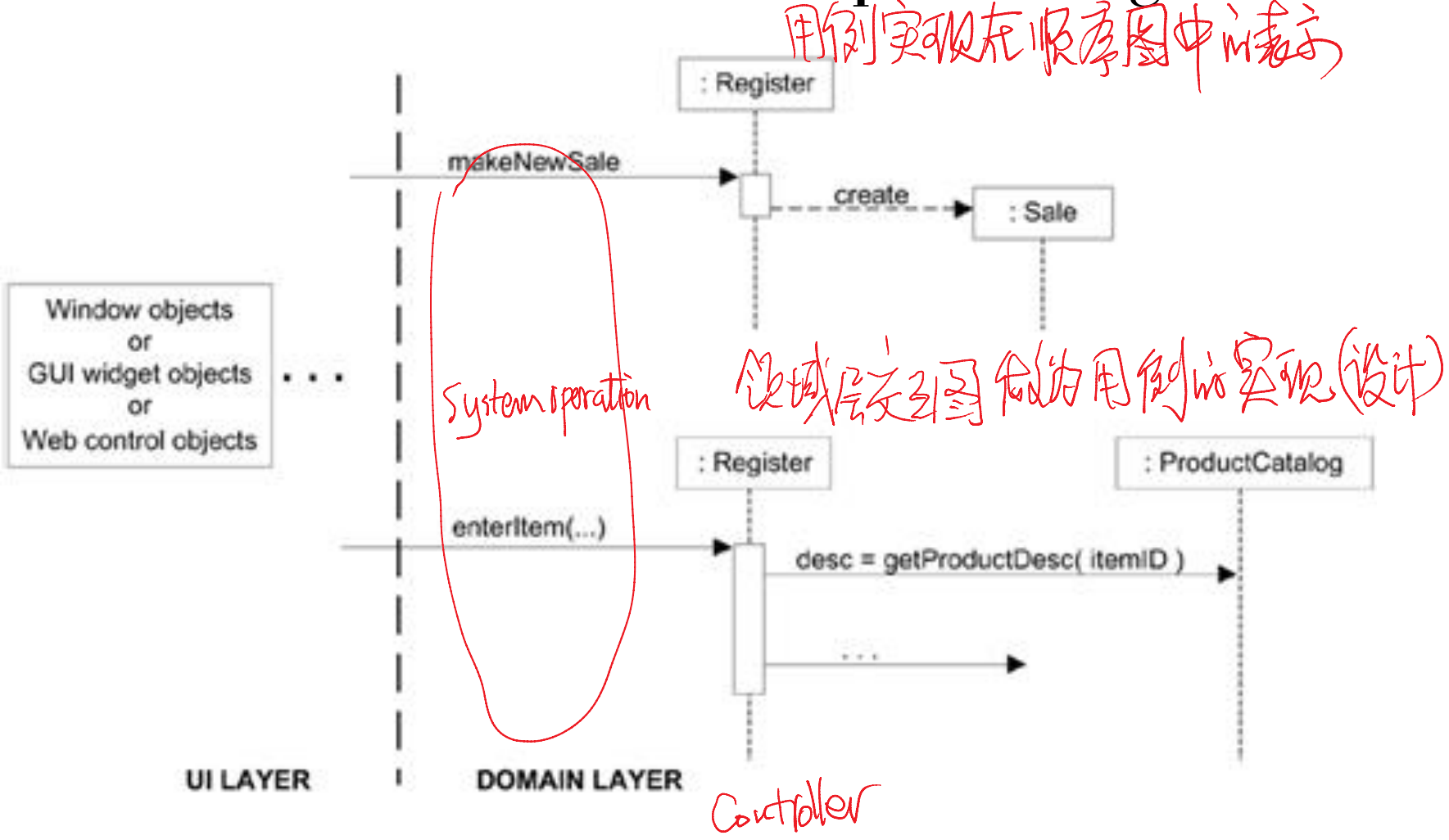
SSD and Use Case Realization

- The use case suggests the system operations that are shown in SSDs.
- The system operations become the starting messages entering the Controllers for domain layer interaction diagrams. 系统操作作为Controller的启动消息
  - This is a ***key point*** often missed by those new to OOA/D modeling.
- Domain layer interaction diagrams illustrate how objects interact to fulfill the required tasks - the use case realization 领域层交互图表示用例实现

Use Case Realizations
System Operation Handling



Use Case Realizations in Sequence Diagram



Key Point: The system operations in the SSDs are used as the starting messages into the domain layer controller objects.

Use Cases and Use Case Realizations

- Naturally, use cases are a prime input to use case realizations.
- The use case text and related requirements expressed in the Supplementary Specifications, Glossary, UI prototypes, report prototypes, and so forth, all inform developers what needs to be built.
- But bear in mind that written requirements are imperfect, often very imperfect.
- (agile) Business people and developers must work together daily throughout the project

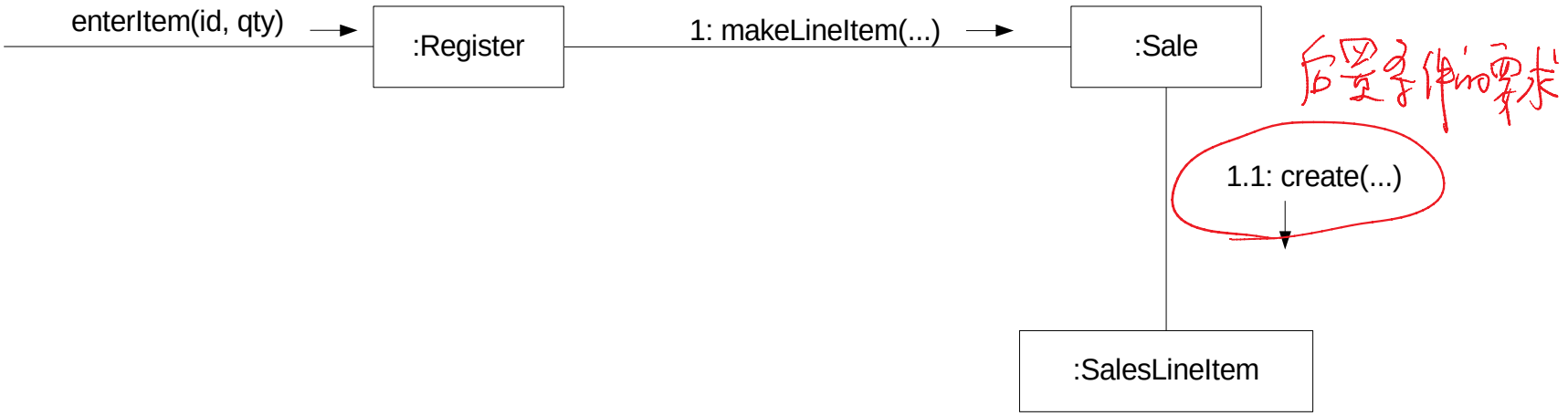
Operation Contract and Use Case Realization

- Use case realizations could be designed directly from the use case text or from one's domain knowledge.
- For some complex system operations, contracts may add more analysis detail.

Contract CO2: enterItem

Operation:	enterItem(itemID : ItemID, quantity : integer)
Cross References:	Use Cases: Process Sale
Preconditions:	There is a sale underway.
Postconditions:	- A SalesLineItem instance sli was created (instance creation).

Operation Contract and Use Case Realization



- In conjunction with contemplating the use case text, for each contract, we work through the postcondition state changes and design message interactions to satisfy the requirements

Domain Model and Use Case Realizations

- In the interaction diagrams, the Domain Model inspires some of the software objects
- The existing Domain Model, as with all analysis artifacts, won't be perfect; you should expect errors and omissions.
- You will discover new concepts previously missed
- Ignore concepts previously identified
- Do likewise with associations and attributes.
- It's normal to discover new conceptual classes during design work
- And to make up software classes whose names and purpose are completely unrelated to the Domain Model.

用例实现之作

Use Case Realization

- Look at the objects suggested by the use case
- Determine who should create them
- Decide how they should interact
- Determine which patterns are being used.

我用到中的对象，
我它的 Creator
确定它们之间的交互.
确定所采用的模式.

下面以例说明GRASP模式的应用.

Use Case Realization with GRASP Patterns

- The following explores the choices and decisions made during the design of a use case realization with objects based on the GRASP patterns.
  - Start Up Use Case
  - Process Sale Use Case

以NextGen POS Case study 讨论 Process Sale 的用例实现

Guideline

- When coding, program at least some Start Up initialization first. 程序编码时，至少要先做一些启动初始化.
- But during OO design modeling, consider the Start Up initialization design *last*, after you have discovered what really needs to be created and initialized. 在建模时，最后再做启动初始化.
- Then, design the initialization to support the needs of other use case realizations

然后再考虑其它用例实现所需的初始化.

- Based on this guideline, we will explore the *Process Sale* use case realization before the supporting *Start Up* design.

我们先讲述 ProcessSale，再讲述 Start Up.

我们来详细设计makeNewSale开始：（用例Process Sale的操作）

Design *makeNewSale*

The *makeNewSale* system operation occurs when a cashier initiates a request to start a new sale, after a customer has arrived with things to buy.

操作约定.

Contract CO1: makeNewSale

Operation:	makeNewSale() 操作.
Cross References:	Use Cases: Process Sale 用例
Preconditions:	none
Postconditions:	- A Sale instance s was created (instance creation). - s was associated with the Register (association formed). - Attributes of s were initialized.

后置条件的细节

我们需要先选择Controller对象(类)

Choosing the Controller Class

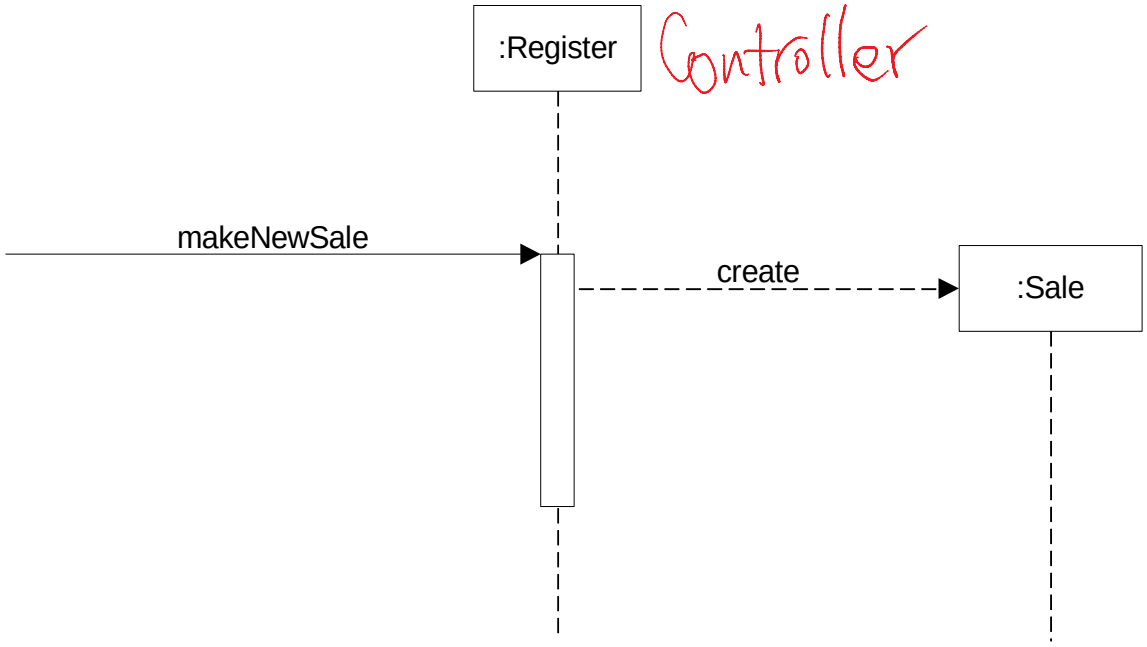
- Our first design choice involves choosing the controller for the system operation message *makeNewSale*.
- By the Controller pattern, here are some choices:

Controller选择原则(两大类)

Represents the overall "system," "root object," a specialized device, or a major subsystem.	**Store**: a kind of root object because we think of most of the other domain objects as "within" the Store. **Register:** a specialized device that the software runs on; also called a POSTerminal. **POSSystem**: a name suggesting the overall system
Represents a receiver or handler of all system events of a use case scenario.	**ProcessSaleHandler:** constructed from the pattern <use-case-name> "Handler" or "Session" **ProcessSaleSession**

Facade

Handler



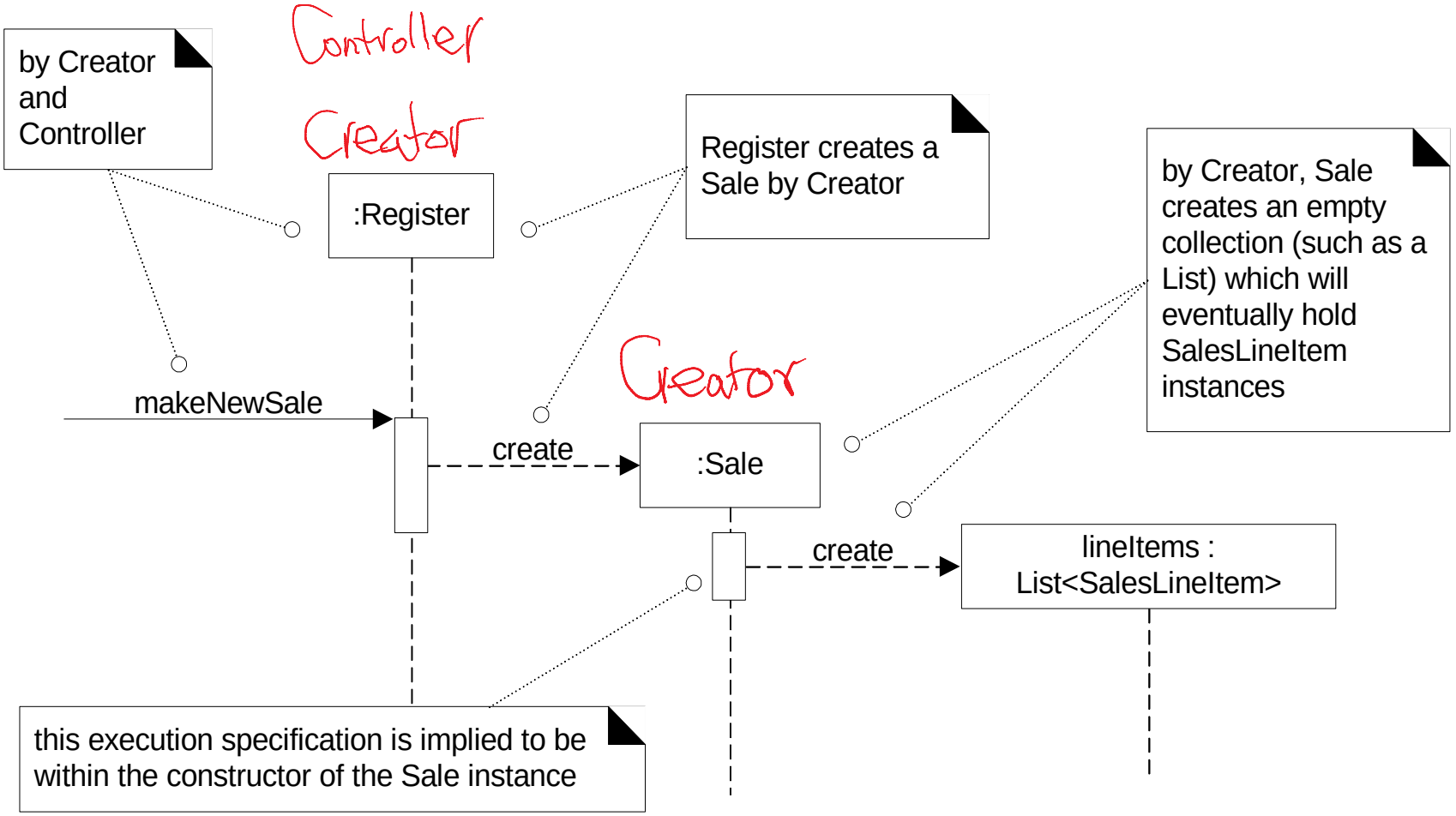
Applying the GRASP Controller pattern

- Choosing a device-object facade controller like *Register*
  - if there are only a few system operations and
  - if the facade controller is not taking on too many responsibilities (not becoming incohesive).
- Choosing a use case controller when
  - we have many system operations and
  - we wish to distribute responsibilities in order to keep each controller class lightweight and focused (cohesive).
- In this case, *Register* suffices since there are only a few system operations.

Creating a New Sale

- We must create a software Sale object
- Creator suggests assigning the creation to a class that aggregates, contains, or records the object to be created.
- Register may be thought of as recording a Sale, Thus, Register is a reasonable candidate for creating a Sale.
- By having the Register create the Sale, during future operations within the session, the Register will have a reference to the current Sale instance.
- When the Sale is created, it must create an empty collection to record all the *SalesLineItem* instances that will be added.

Sale and Collection Creation



现在讨论如何设计enterItem.

Design *enterItem*

We now construct an interaction diagram to satisfy the postconditions of *enterItem*, using the GRASP patterns to help with the design decisions

Contract CO2: enterItem

操作约定

Operation:	enterItem(itemID : ItemID, quantity : integer) 参数
Cross References:	Use Cases: Process Sale 用例
Preconditions:	There is an underway sale.
Postconditions:	- A SalesLineItem instance sli was created (instance creation). 实例创建 - sli was associated with the current Sale (association formed). 形成关联 - sli.quantity became quantity (attribute modification). 状态修改 - sli was associated with a ProductDescription, based on itemID match (association formed). 形成关联

Choosing the Controller Class

先选控制类

- Who can handle the responsibility for the system operation message *enterItem*.
- Based on the Controller pattern, as for *makeNewSale*, we will continue to use Register as a controller.

我们仍选Register作为Controller

Display Item Description and Price?

- Because of a principle of Model-View Separation, it is not the responsibility of non-GUI objects (such as a Register or Sale) to get involved in output tasks.
- Therefore, although the use case states that the description and price are displayed after this operation, we ignore the design at this time.
- All that is required with respect to responsibilities for the display of information is that the information is known, which it is in this case.

Creating a New SalesLineItem

- The enterItem contract postconditions indicate the creation, initialization, and association of a *SalesLineItem*.
- Who create?
  - Domain Model reveals that a Sale contains SalesLineItem objects.
  - We determine that a software Sale may similarly contain software SalesLineItem.
- The postconditions indicate that the new SalesLineItem needs a quantity when created
  - therefore, the Register must pass it along to the Sale, which must pass it along as a parameter in the create message.
  - In Java, that would be implemented as a constructor call with a parameter.

Creating a New SalesLineItem

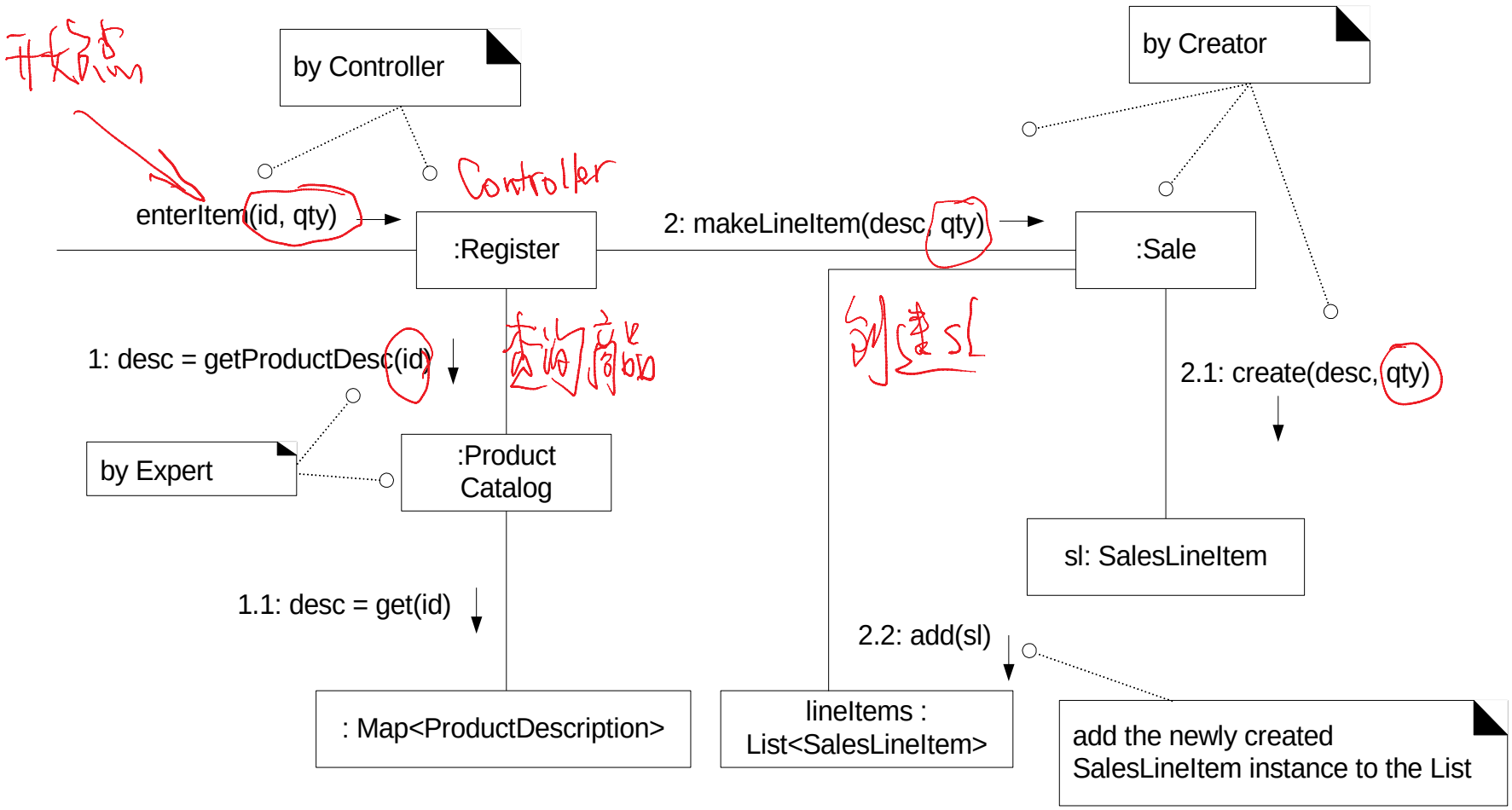
- Therefore, by Creator, a makeLineItem message is sent to a Sale for it to create a SalesLineItem.
- The Sale creates a SalesLineItem, and then stores the new instance in its permanent collection.
- The parameters to the makeLineItem message include the quantity, so that the SalesLineItem can record it.

Other Design Issues

- Who should be responsible for knowing a ProductDescription, based on an itemID match?
  - *ProductCatalog* logically contains all the *ProductDescriptions*.
- Who should send the getProductDescription message to the ProductCatalog to ask for a ProductDescription
  - For an object to send a message to another object, it must have visibility to it.

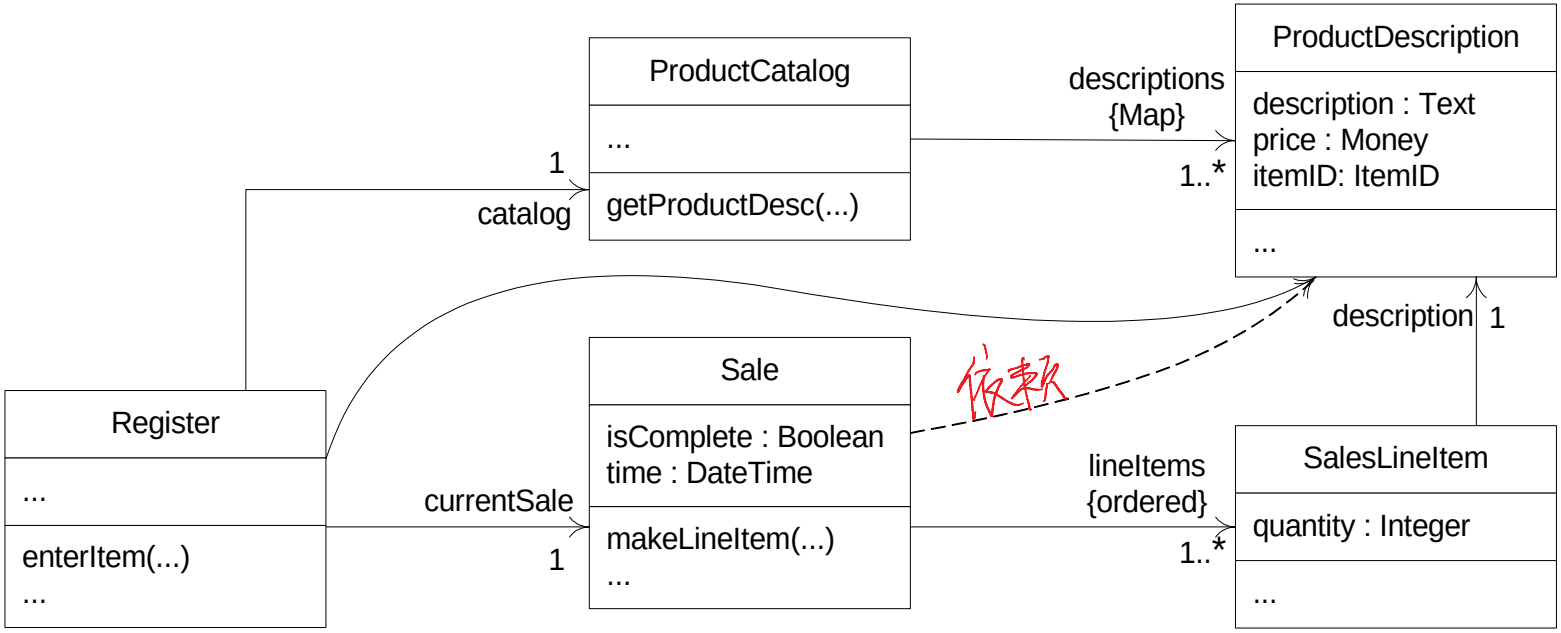
The *enterItem* interaction diagram
Dynamic view

Final design



设计依据是 enterItem 的 postcondition.

Partial DCD related to the *enterItem* design
Static view



与enterItem有关的设计类图(部分)

Design *endSale*

Contract CO3: endSale

Operation:	endSale()
Cross References:	Use Cases: Process Sale
Preconditions:	There is an underway sale.
Postconditions:	Sale.isComplete became true (attribute modification).

- Modeling Steps
  - Choose the controller class
  - Setting the *Sale.isComplete* Attribute
  - Calculating the Sale Total
  - Designing *Sale.getTotal*

Choosing the Controller Class

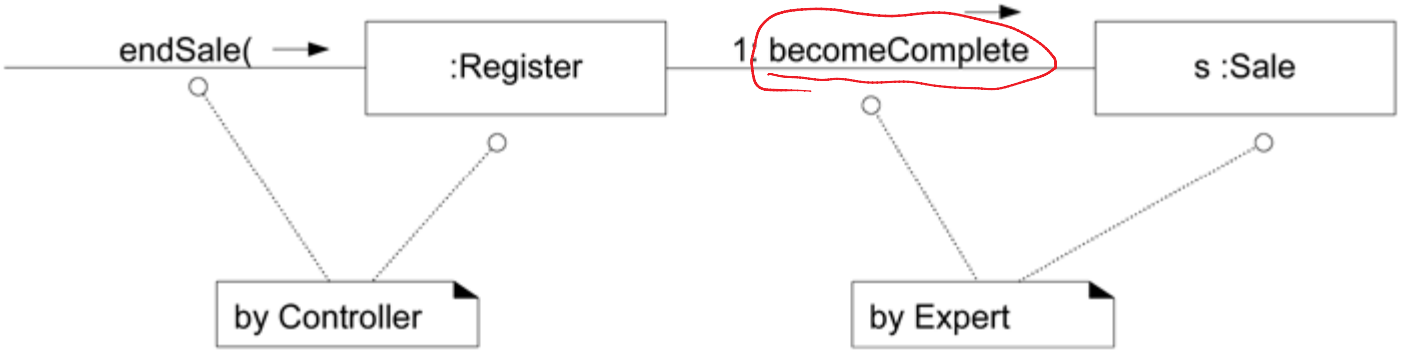
- Based on the Controller GRASP pattern, as for *enterItem*, we will continue to use *Register* as a controller.

同前，仍选Register为Controller

Completion of item entry

- The contract postconditions state:

  Sale.isComplete became *true* (attribute modification).
- Who should be responsible for setting the *isComplete* attribute of the *Sale* to true?
- By Expert, it should be the *Sale* itself, since it owns and maintains the *isComplete* attribute.
- Thus, the *Register* will send a *becomeComplete* message to the *Sale* to set it to *true*



Calculating the Sale Total

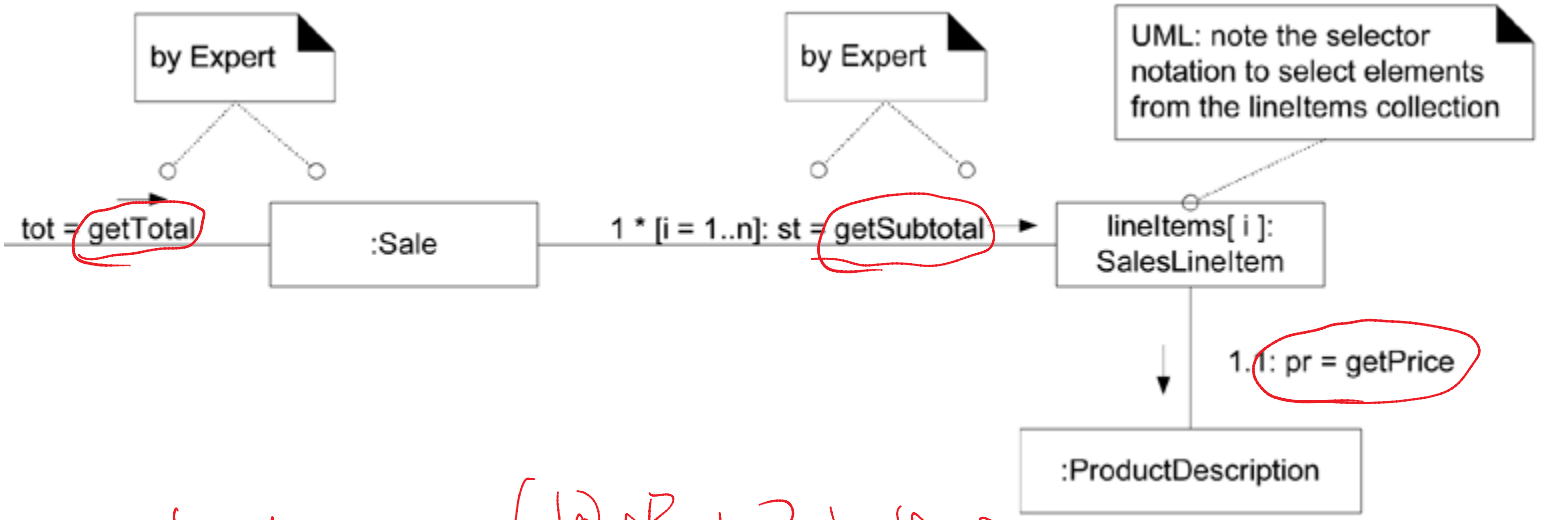
- *Sale* will have the responsibility of knowing its total, implemented as a *getTotal* method.
- *SalesLineItem* will have the responsibility of knowing its subtotal, implemented as a *getSubtotal* method
- *ProductDescription* will have the responsibility of knowing its price, implemented as a *getPrice* operation.

All by Expert.

Information Required for Sale Total	Information Expert
ProductDescription.price	*ProductDescription*
SalesLineItem.quantity	*SalesLineItem*
all the *SalesLineItems* in the current Sale	*Sale*

前面曾经讲过这个例子

Sale.getTotal Interaction Diagram



=> Message (根据上页ppt选择)

这个设计在前面做为例子出现过。

Showing a Method in a Note Symbol

现在讨论 makePayment 的设计

Design *makePayment*

Contract CO4: makePayment

Operation:	makePayment(amount: Money)
Cross References:	Use Cases: Process Sale 用例
Preconditions:	There is an underway sale.
Postconditions:	- A Payment instance p was created (instance creation). - p.amountTendered became amount (attribute modification). - p was associated with the current Sale (association formed). - The current Sale was associated with the Store (association formed); (to add it to the historical log of completed sales).

后置条件

日志需求.

We construct a design to satisfy the postconditions of *makePayment*.

我们的设计要满足 Contract 的 Postconditions.

Design *makePayment*

- Creating the Payment
- Logging a Sale
- Calculating the Balance

设计工作.

Creating the Payment

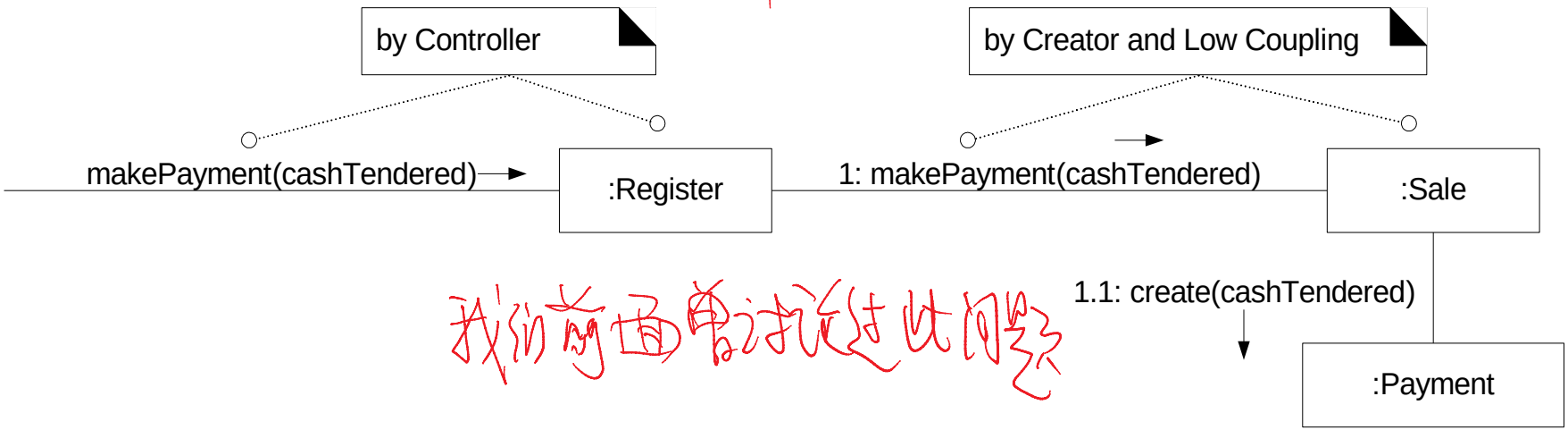
There are two candidates for creating Payment: *Register* (by Expert), *Sale* (by Creator)

(自己分析，前面常讨论过)

Guideline:

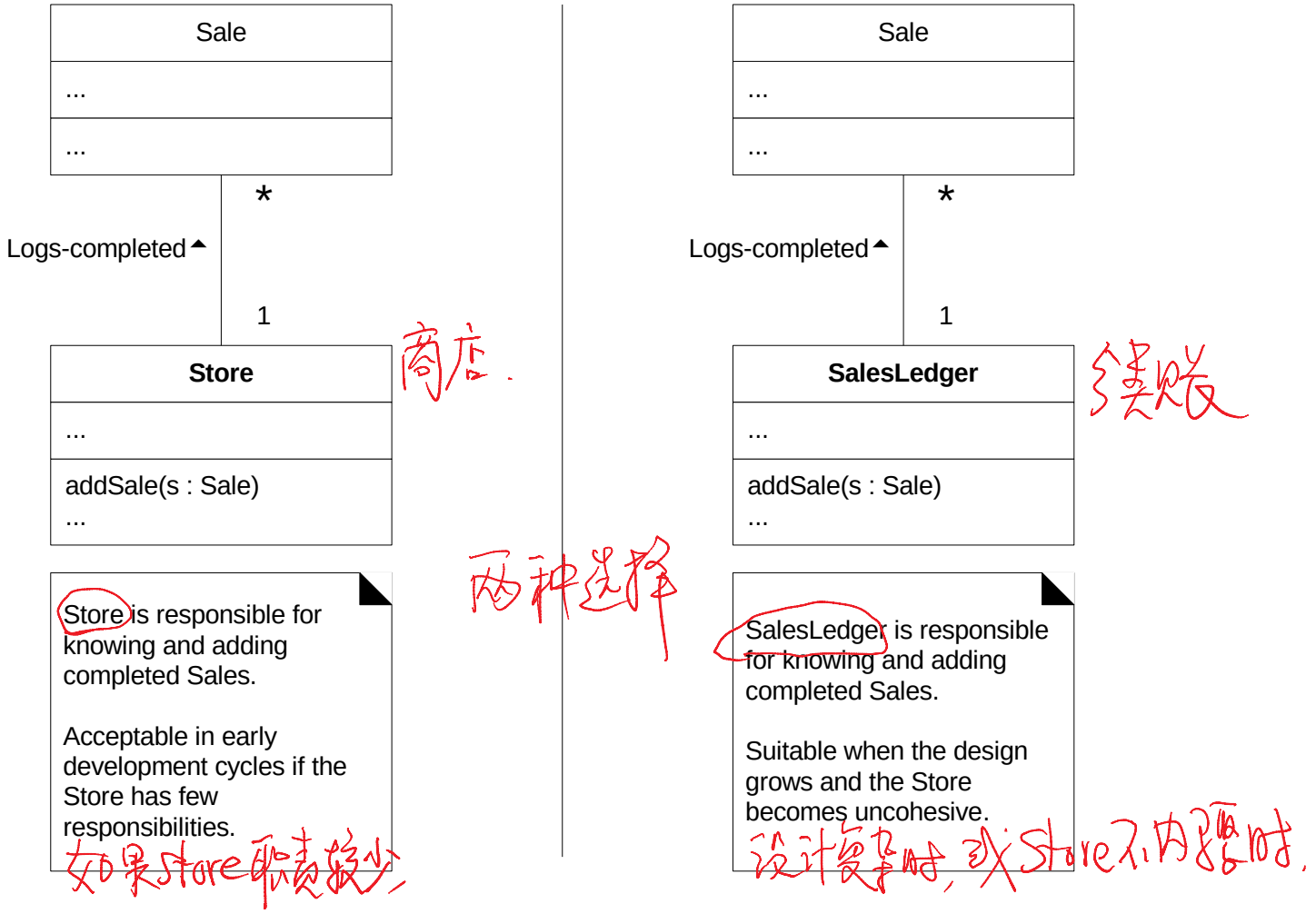
When there are alternative design choices, take a closer look at the **cohesion and coupling** implications of the alternatives, and possibly at the future evolution pressures on the alternatives. Choose an alternative with good cohesion, coupling, and stability in the presence of likely future changes

我们选择Sale为Creator



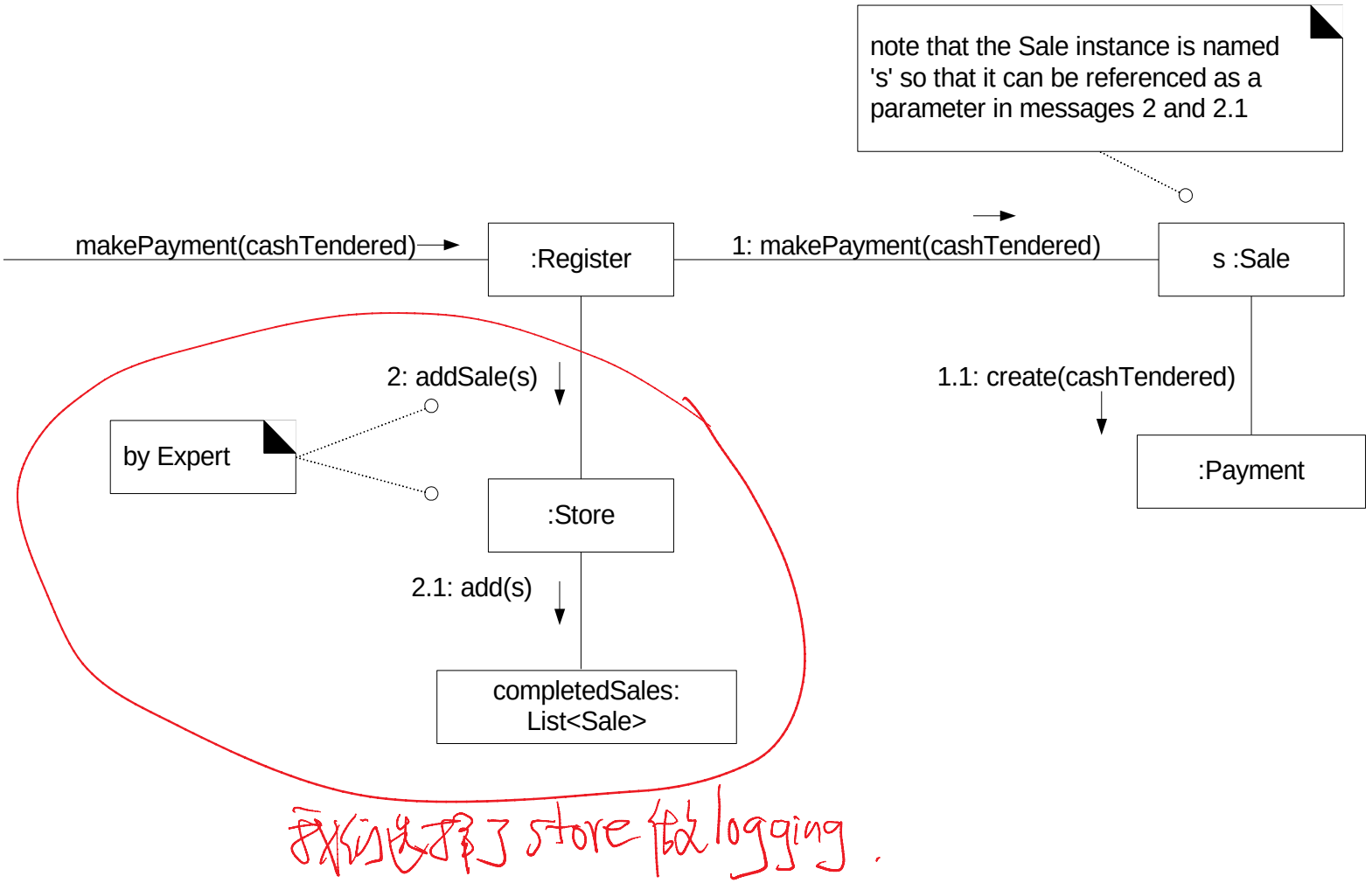
我们前面常讨论这种关系

Who should be responsible for knowing the completed sales?



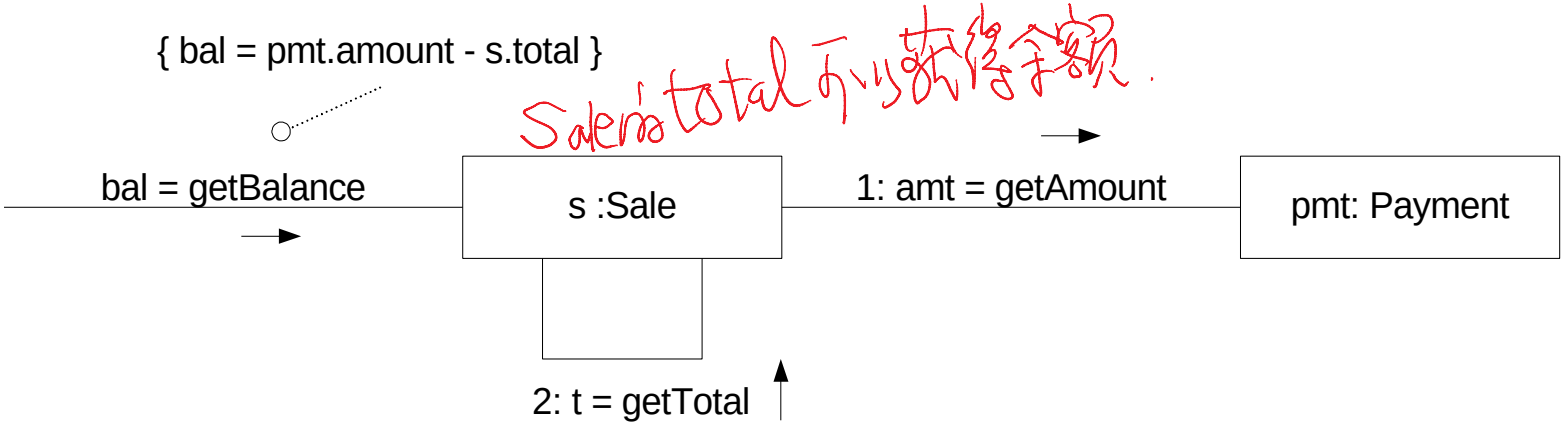
Who is responsible for knowing all the logged sales and doing the logging?

Logging a Completed Sale

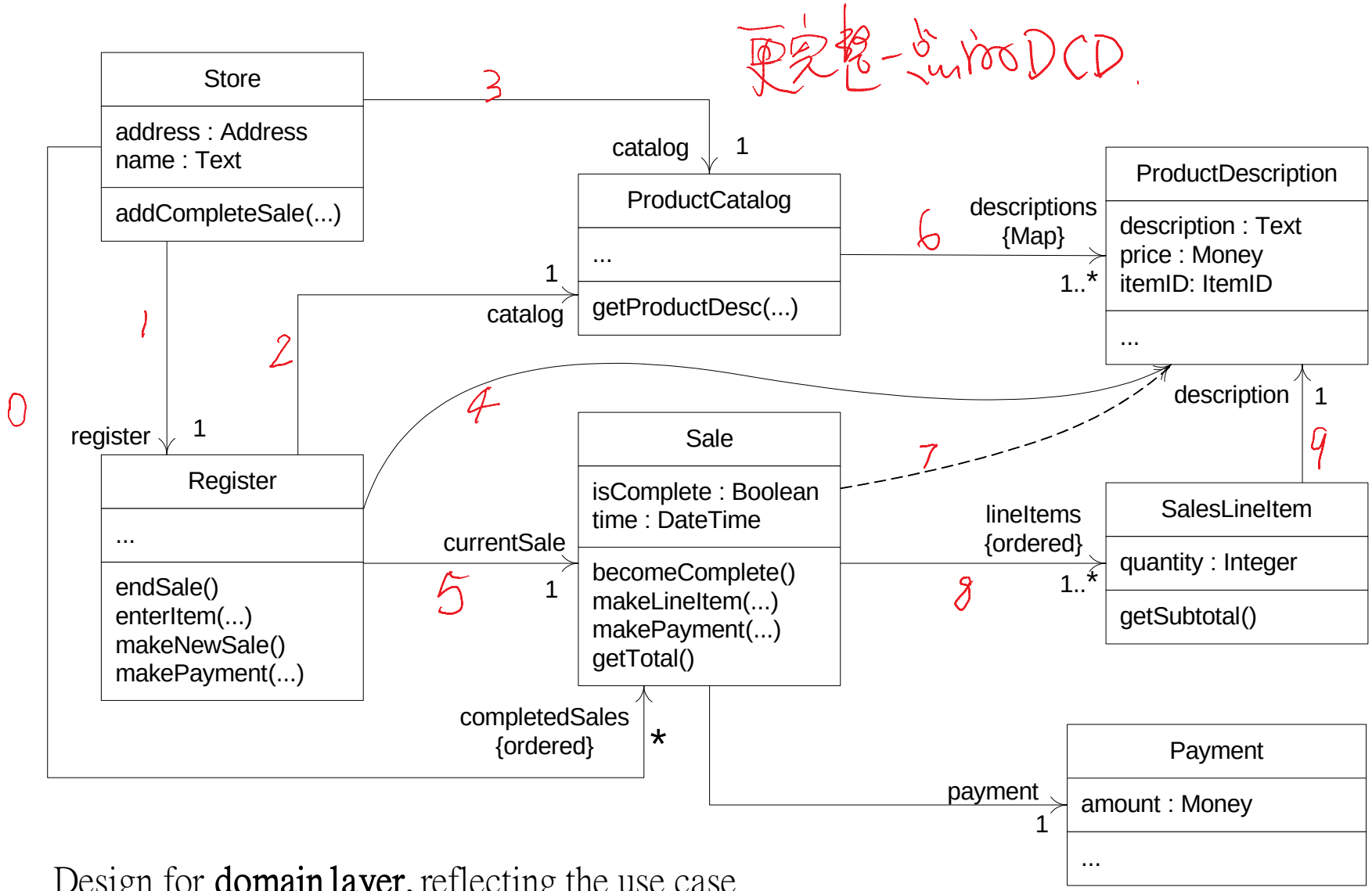


Sale.getBalance interaction diagram

- The *Process Sale* use case implies that the balance due from a payment be printed on a receipt and displayed somehow.



A more complete DCD reflecting most design decisions

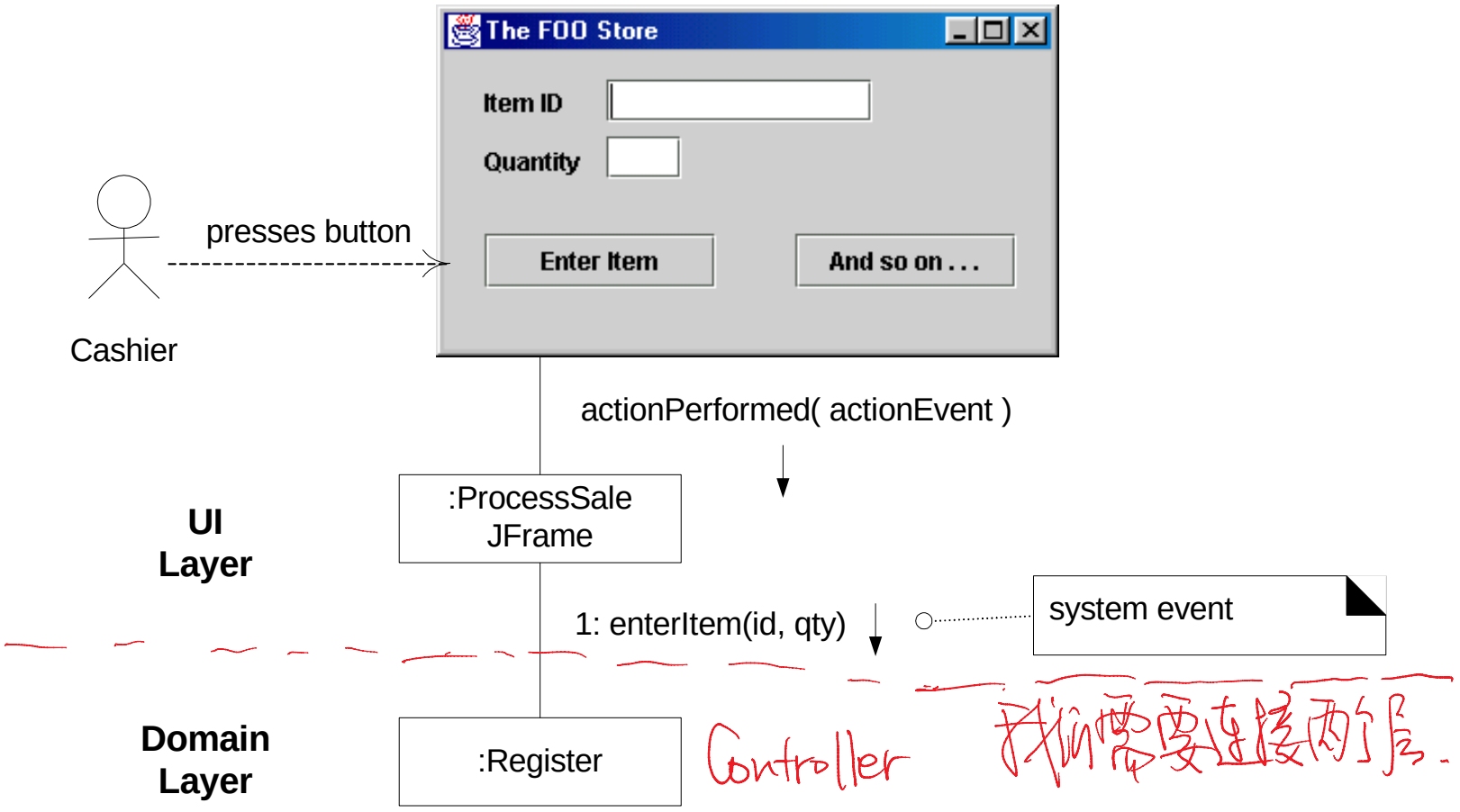


Design for **domain layer**, reflecting the use case realizations for *Process Sale* in iteration-1.

Connect the UI Layer to the Domain Layer

- We still have more OO design work to do in other layers, include the UI layer and services layers.
- Common designs by which objects in the UI layer obtain visibility to objects in the domain layer include the following:
  - An *initializer* object called from the application starting method (e.g., the Java main method) creates both a UI and a domain object and passes the domain object to the UI.
  - A UI object retrieves the domain object from a well-known source, such as a factory object that is responsible for creating domain objects.
- Once the UI object has a connection to the Register instance (the facade controller in this design), it can forward system event messages, such as the *enterItem* and *endSale* message, to it

Connect the UI Layer to the Domain Layer

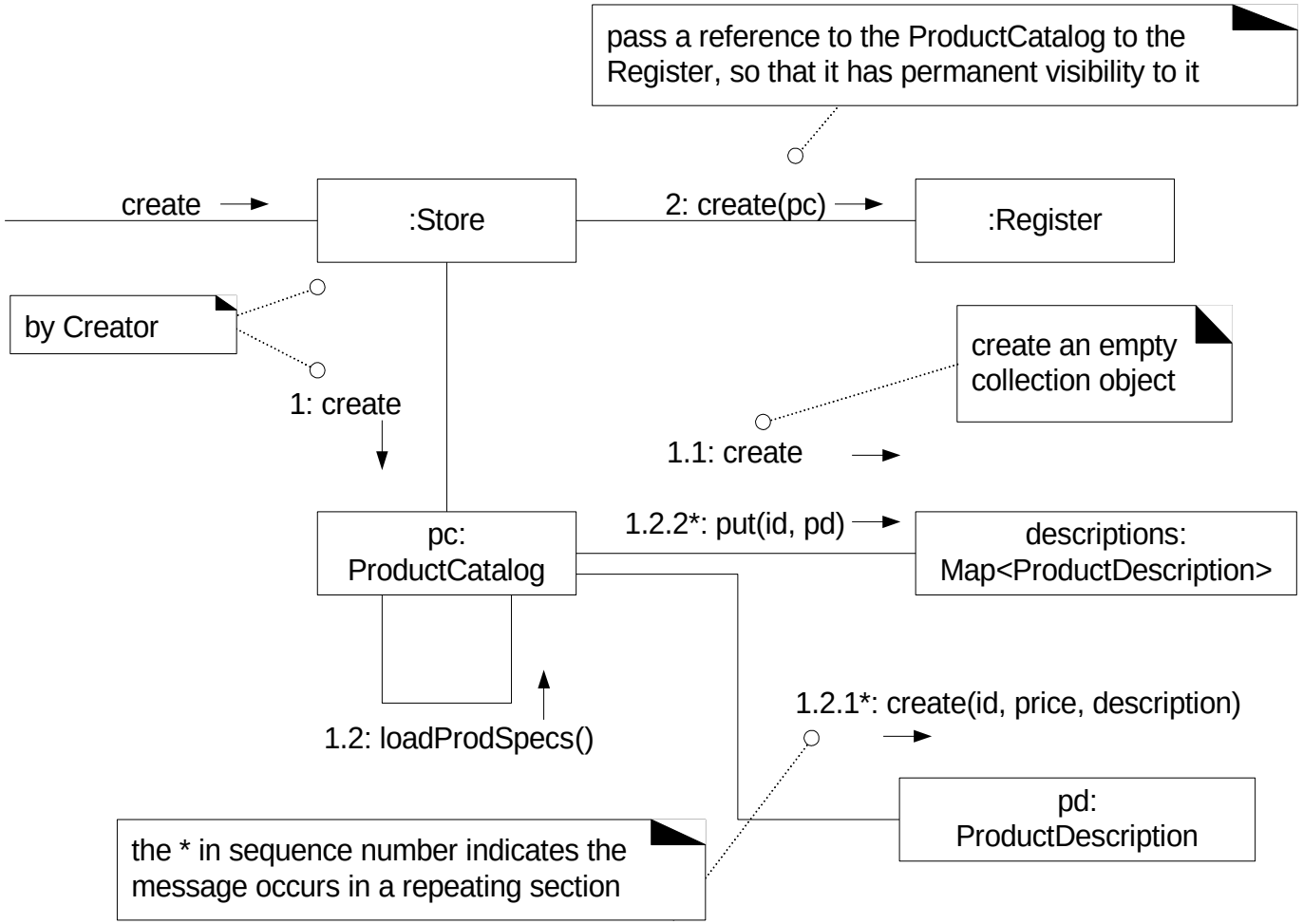


Initialization and the 'Start Up' Use Case

- When to Create the Initialization Design?
  - **Guideline:** Do the initialization design last.
- How do Applications Start Up?

```
public class Main {
   public static void main( String[] args ) {
   // Store is the initial domain object.
   // The Store creates some other domain objects.
   Store store = new Store();
   Register register = store.getRegister();
   ProcessSaleJFrame frame = new ProcessSaleJFrame( register ); ...
   }
}
```

Creation of the initial domain object and subsequent objects



各个对象的创建及初始化过程.

最后说明对象设计的迭代和进化方法.

Iterative and Evolutionary Object Design

- Many suggestions about iterative and evolutionary object design for use case realizations over the last few chapters
- The essential point: 基本要点:
  - Keep it light and short, move quickly to code and test, and don't try to detail everything in UML models.
  - Model the creative, difficult parts of the design.

前面已有很多讨论

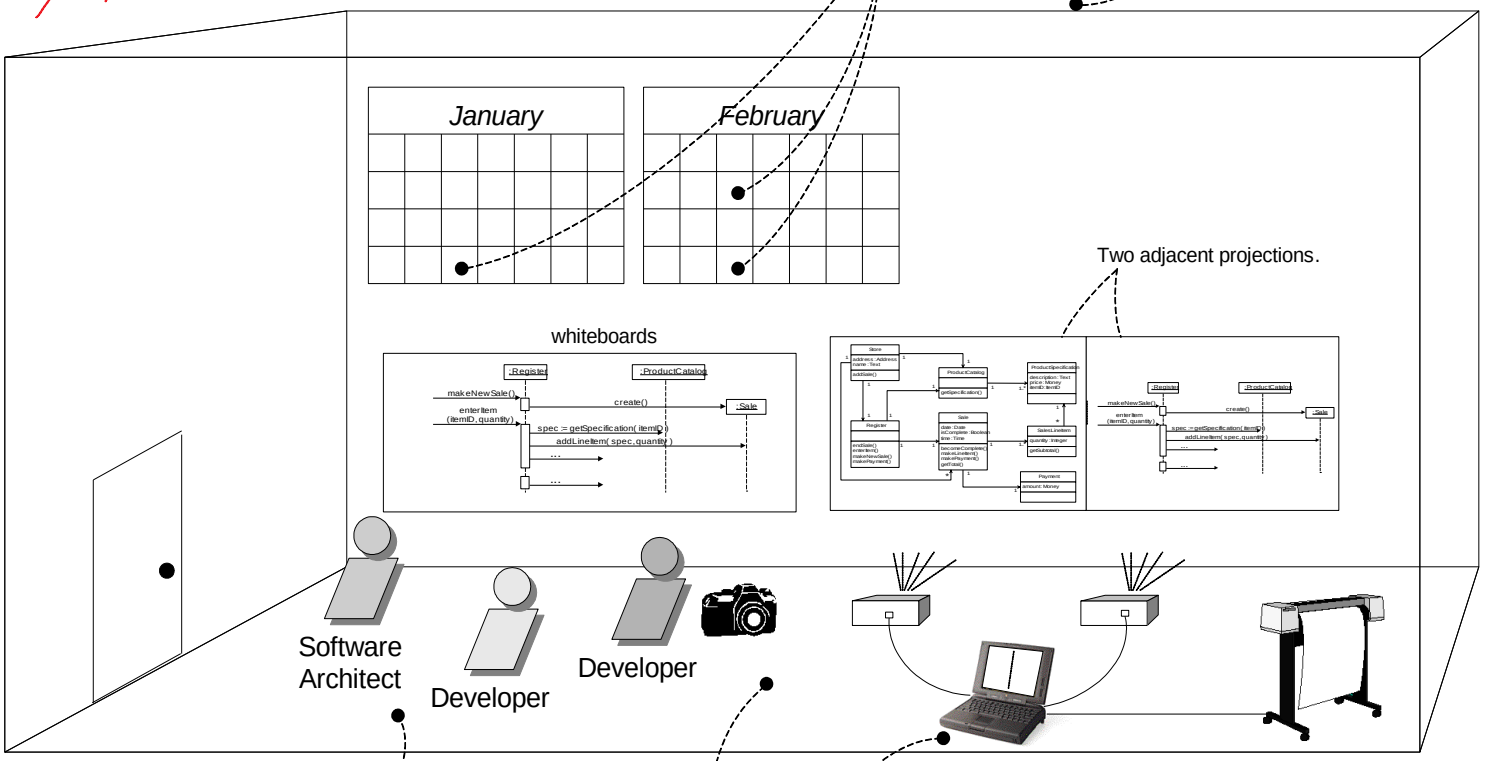
Sample Process and Setting Context

When

Near the beginning of each iteration, for a "short" period before programming.

Where

In a project room with lots of support for drawing and viewing drawings.



Who

Perhaps developers will do some design work in pairs. The software architect will collaborate, mentor, and visit with different design groups.

How: Tools

Software: A UML CASE tool that can also reverse engineer diagrams from code.

Hardware:

- Use two projectors attached to dual video cards.
- For whiteboard drawings, perhaps a digital camera.
- To print noteworthy diagrams for the entire team, a plotter for large-scale drawings to hang on walls.

Object Design Within the UP

- Use case realizations are part of the UP Design Model.
- **Inception** The Design Model and use case realizations will not usually be started until elaboration because they involve detailed design decisions, which are premature during inception. 一般不从Inception开始.
- **Elaboration**
  During this phase, use case realizations may be created for the most architecturally significant or risky scenarios of the design. However, UML diagramming will not be done for every scenario, and not necessarily in complete and fine-grained detail.
  The idea is to do interaction diagrams for the key use case realizations that benefit from some forethought and exploration of alternatives, focusing on the major design decisions.
- **Construction** Use case realizations are created for remaining design problems. 在Construction Phase，仍有一些用例实现工作.

Sample UP Artifacts and Timing

Discipline	Artifact	Incep.	Elab.	Const.	Trans.
	Iteration	I1	E1..En	C1..Cn	T1..T2
Business Modeling	Domain Model		s		
Requirements	Use-Case Model	s	r		
	Vision	s	r		
	Supplementary	s	r		
	Glossary	s	r		
	Business Rules	s	r		
Design	Design Model		s	r	
	SW Architecture		s		
	Data Model		s	r	