COMP 3711

OOD

Use Case Realization
And
GRASP

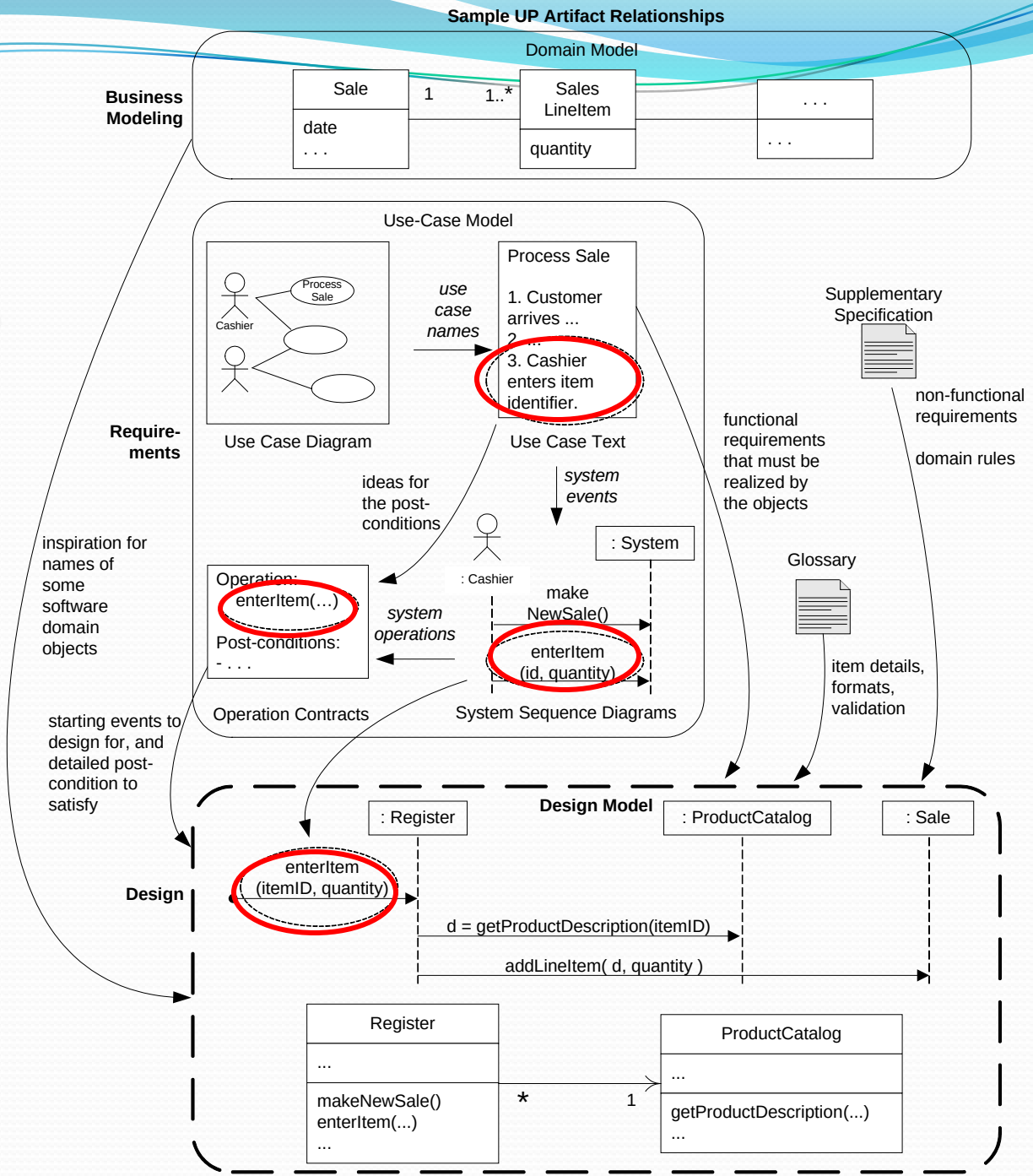
Larman Chapter 18

Use Case Realization

- *"A Use-case realization describes how a particular use case is realized within the Design Model, in terms of collaborating objects" .... RUP*

- Connection between the requirements expressed as use cases and the object design that satisfies the requirements.

OOD Design Artifacts



Larman fig. 18.1

Use Case Realization - Inputs

- Use Case Realization can be designed from:
  - Use case description / use case diagram
  - Operation contracts (e.g. work through post-condition state changes and design message interactions to satisfy requirements)
  - Domain Model (e.g. iterative design that permits inclusion of new conceptual classes that were missed)
  - User (domain experts)

Use Case Realization-Starting Points

- Use Case artifacts suggest the system operations that are shown in SSDs

- System operations are the starting messages entering the Controllers for the domain layer interaction diagrams

- Domain layer interaction diagrams shows how objects interact to fulfill the required tasks

Use Case – A Good Starting Point

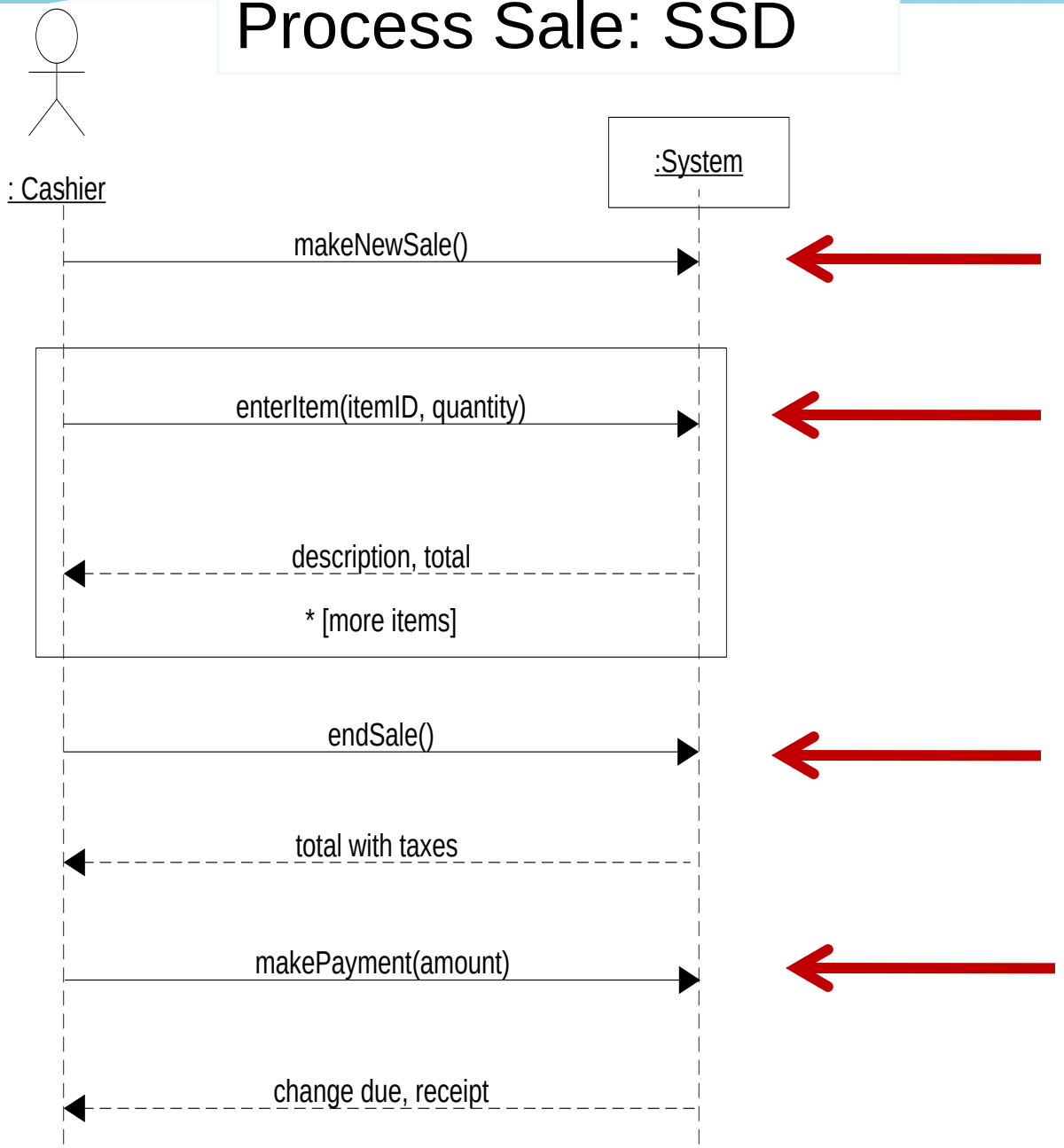
Follow through the NextGen POS example from pp 325-349

NextGen POS Process Sale scenario:

1. Customer arrives at a POS checkout with goods and/or services to purchase.
2. Cashier starts a new sale.
3. Cashier enters item identifier.
   System records sale line item and presents item description, price, and running total.
4. Cashier repeats steps 3-4 until indicates done.
5. System presents total with taxes calculated.
6. Cashier tells Customer the total, and asks for payment.
7. Customer pays and System handles payment.

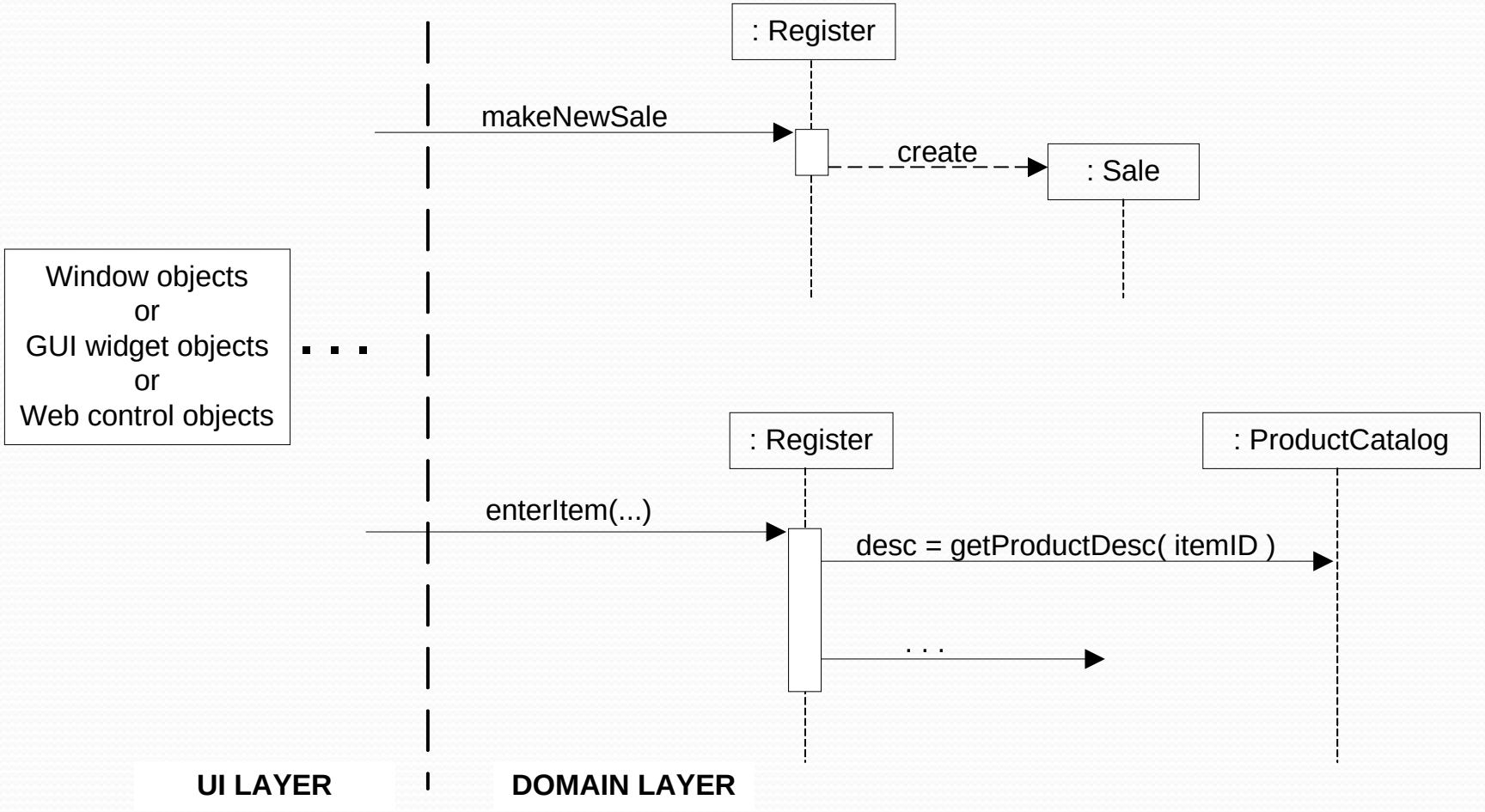
…

Process Sale: SSD



Process Sales – Use Case

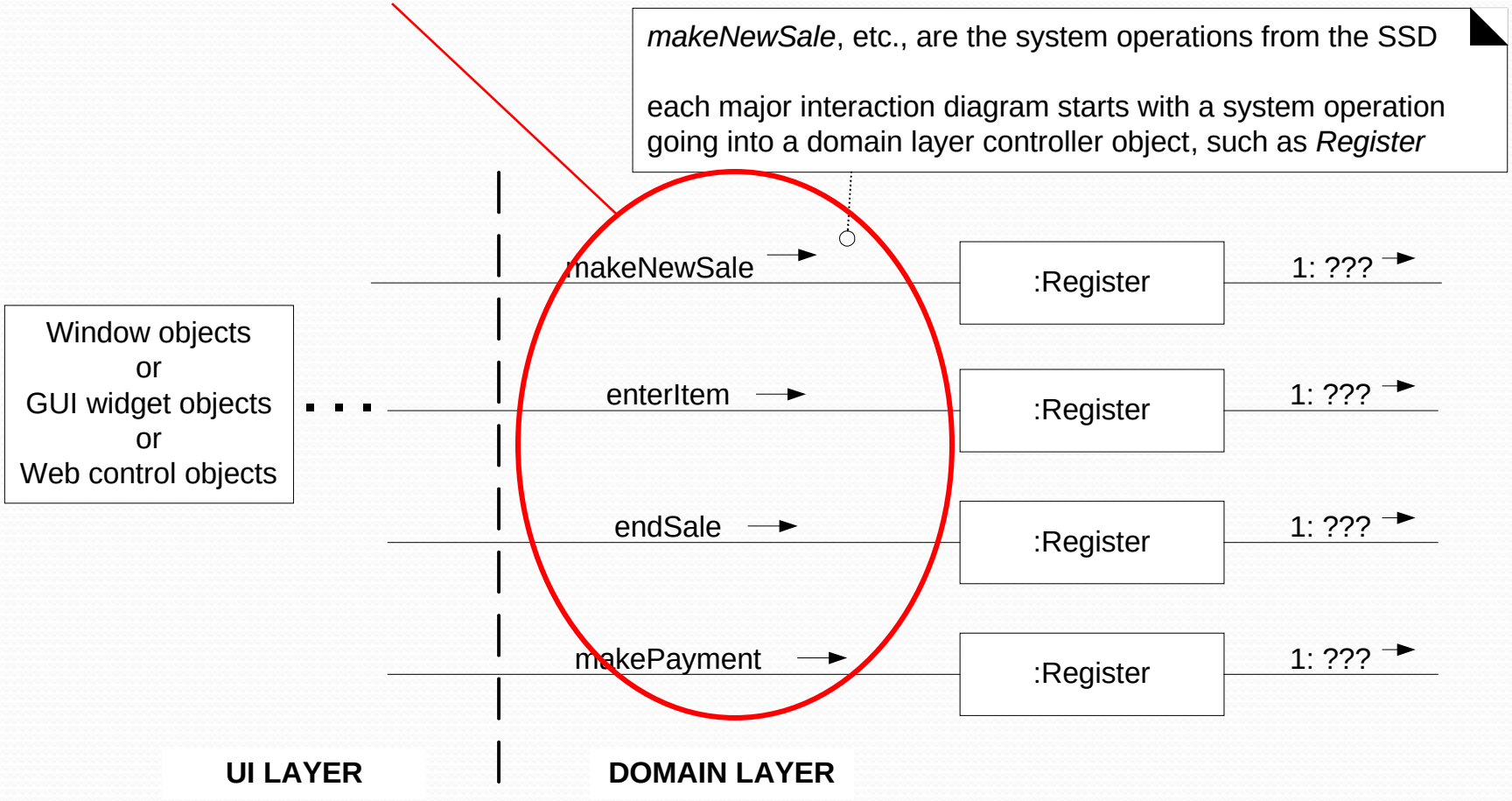
Follow through the NextGen POS example from pp 325-349



Larman fig. 18.3 Sequence Diagram

Process Sales – Use Case

System operations are the starting messages into the domain layer control objects



Larman fig. 18.2 Collaboration (Communication) Diagram

Contract C01 – makeNew Sale

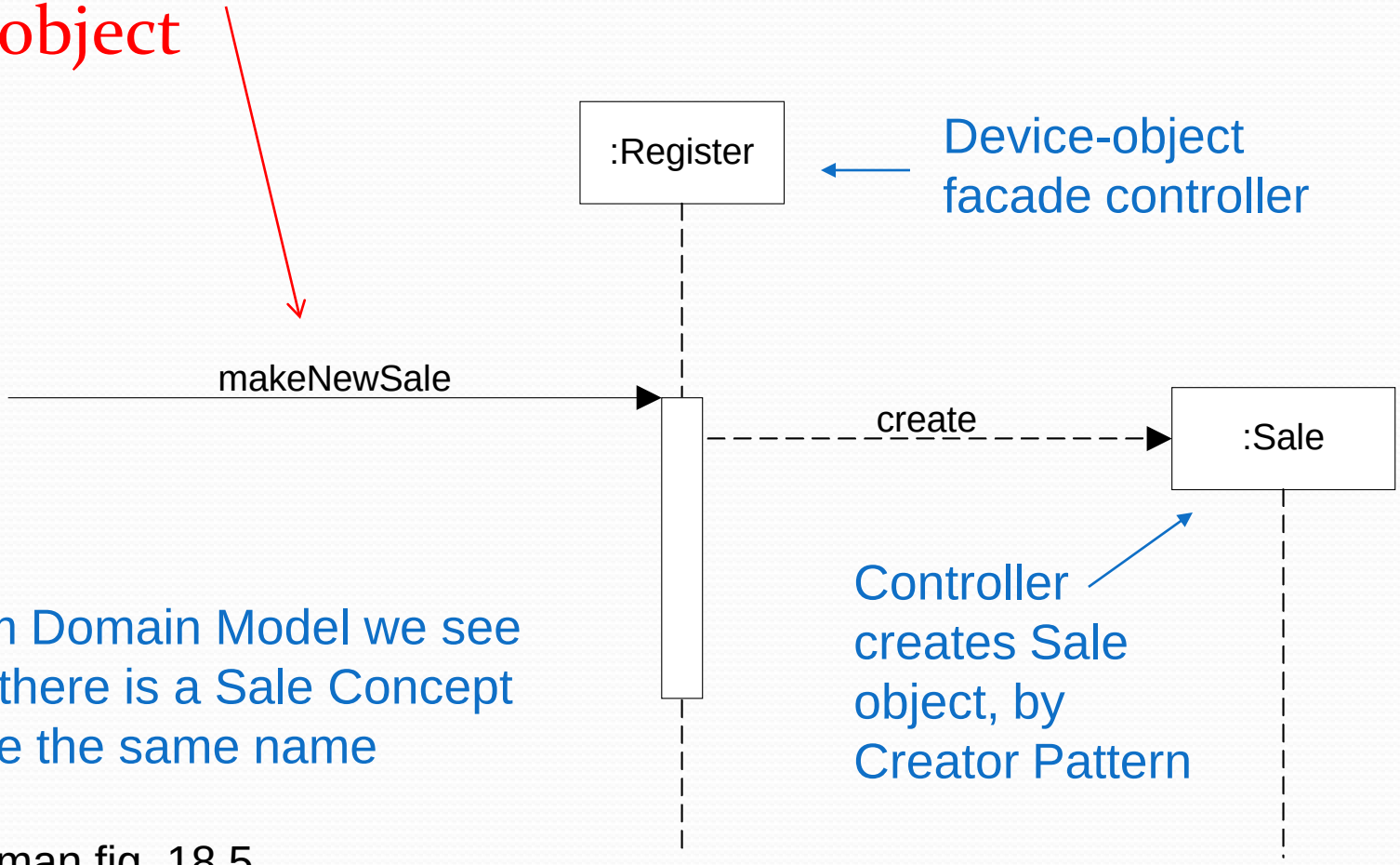
Operation:	makeNewSales
Cross References:	Use Case: Process Sales
Preconditions:	none

Postconditions:

- Sale instance s was created (instance creation)
- s was associated with the Register (association formed)
- Attributes of s were initialized

makeNewSale

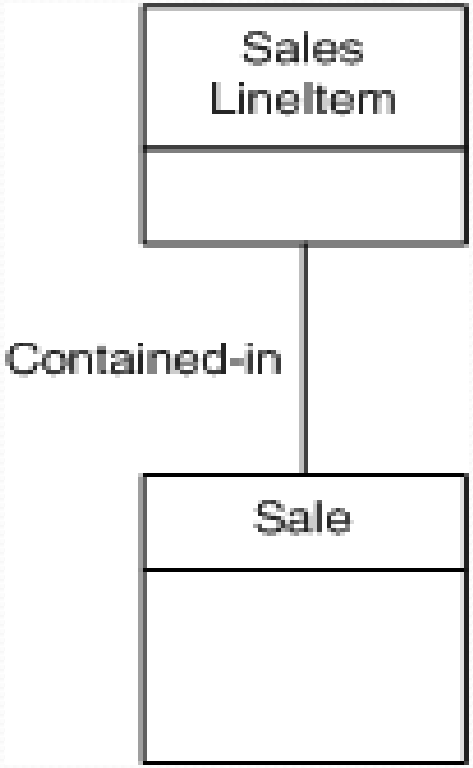
- Based on the Controller pattern, the system operation makeNewSale message is sent to a Register software object



Larman fig. 18.5

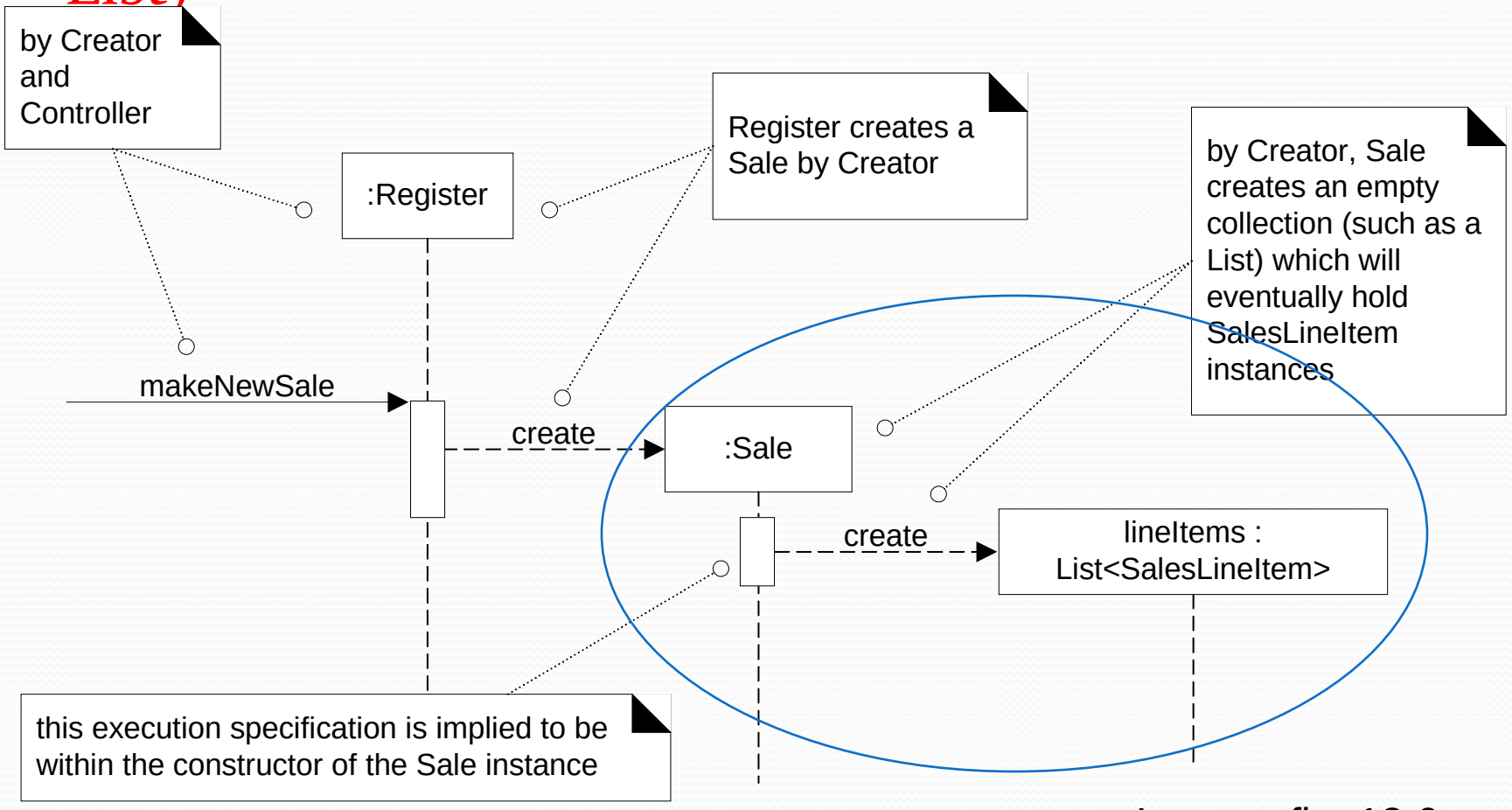
More needed for makeNewSale

From Domain Model sale contains the lineItems



Creators – makeNewSale

- Reasonable to use Register to create the Sale, the Sale creates an empty collection for SalesLineItem(e.g. Java List)



Larman fig. 18.6

Contract C02 – enterItem

Operation:	enterItem(itemID, quantity : integer)
Cross References:	Use Case: Process Sales
Preconditions:	A sale is underway

Postconditions:

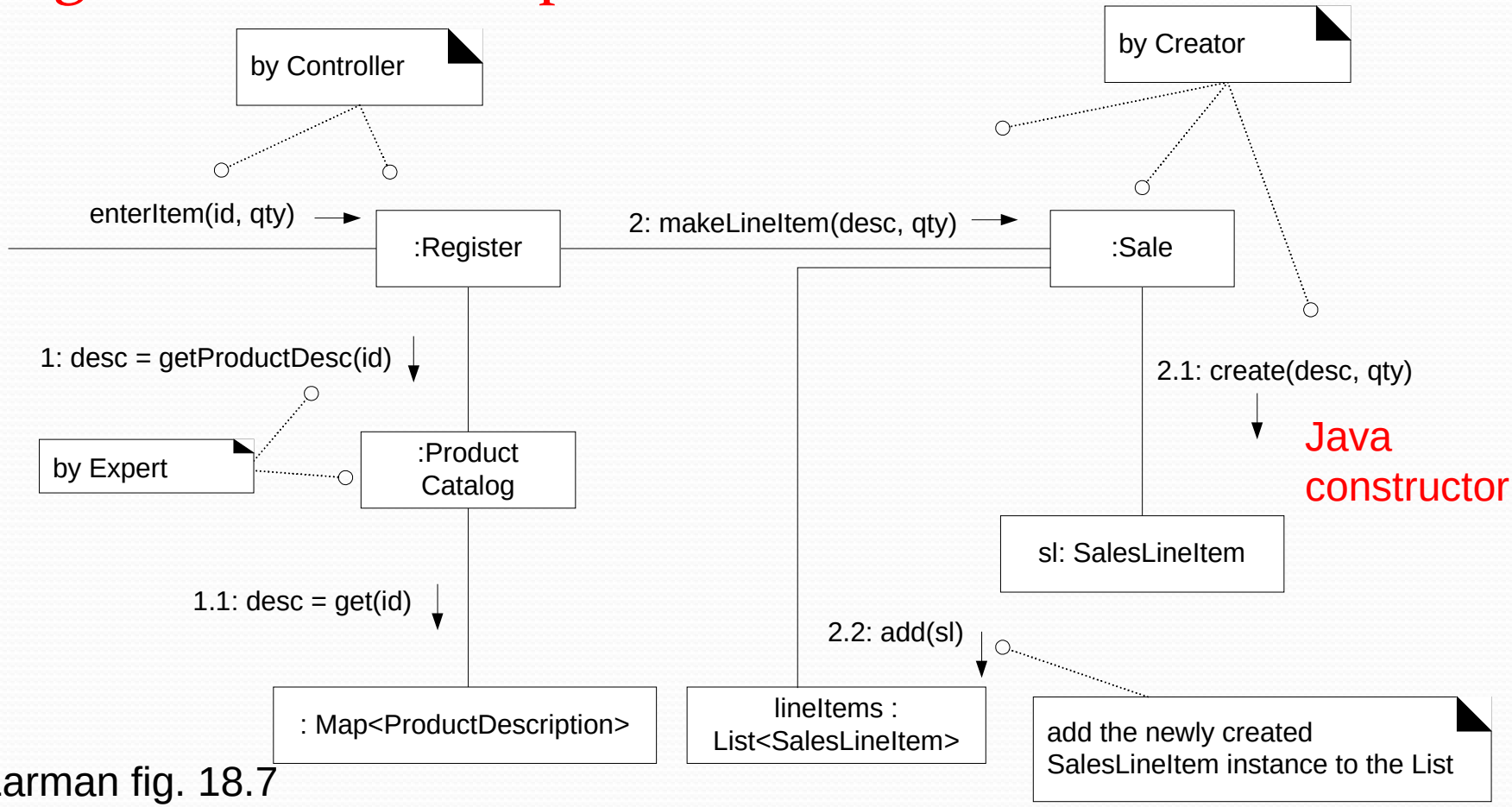
- A SaleLineItem instance sli was created (instance creation)
- sli was associated with the current Sale (association formed)
- sli.quantity became quantity (attribute modification)
- sli was associated with a ProductDescription, base on itemID match (association formed)

A few design decisions - enterItem

- Who should be responsible for displaying the output?
- Who should create the SalesLineItem?
- Who should know the Product detail?

Controller / Creators – enterItem

- Creation, initialization and association of a SalesLineItem with visibility to ProductCatalog to getProductDescription

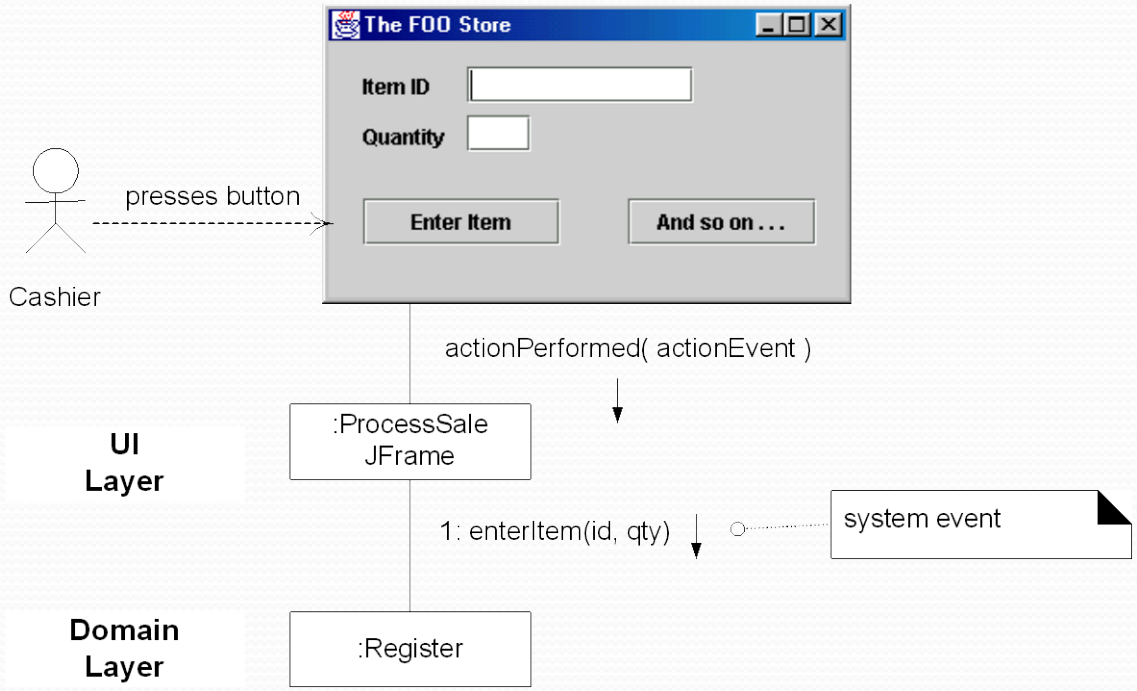


Larman fig. 18.7

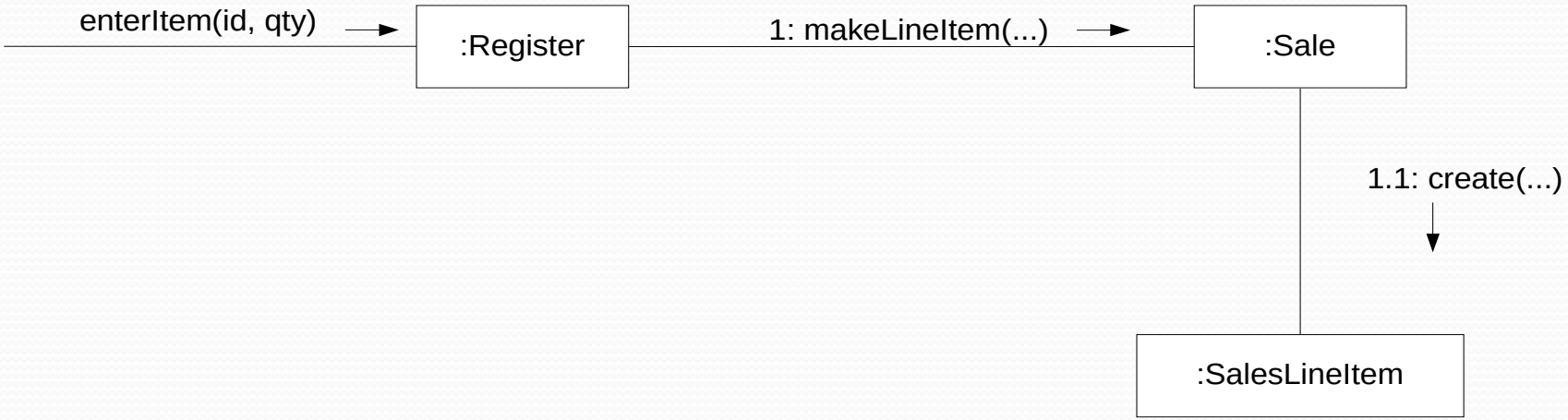
Apply Information Expert Pattern

- 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.

- All that is required regarding responsibilities for the display of information is that the information is known



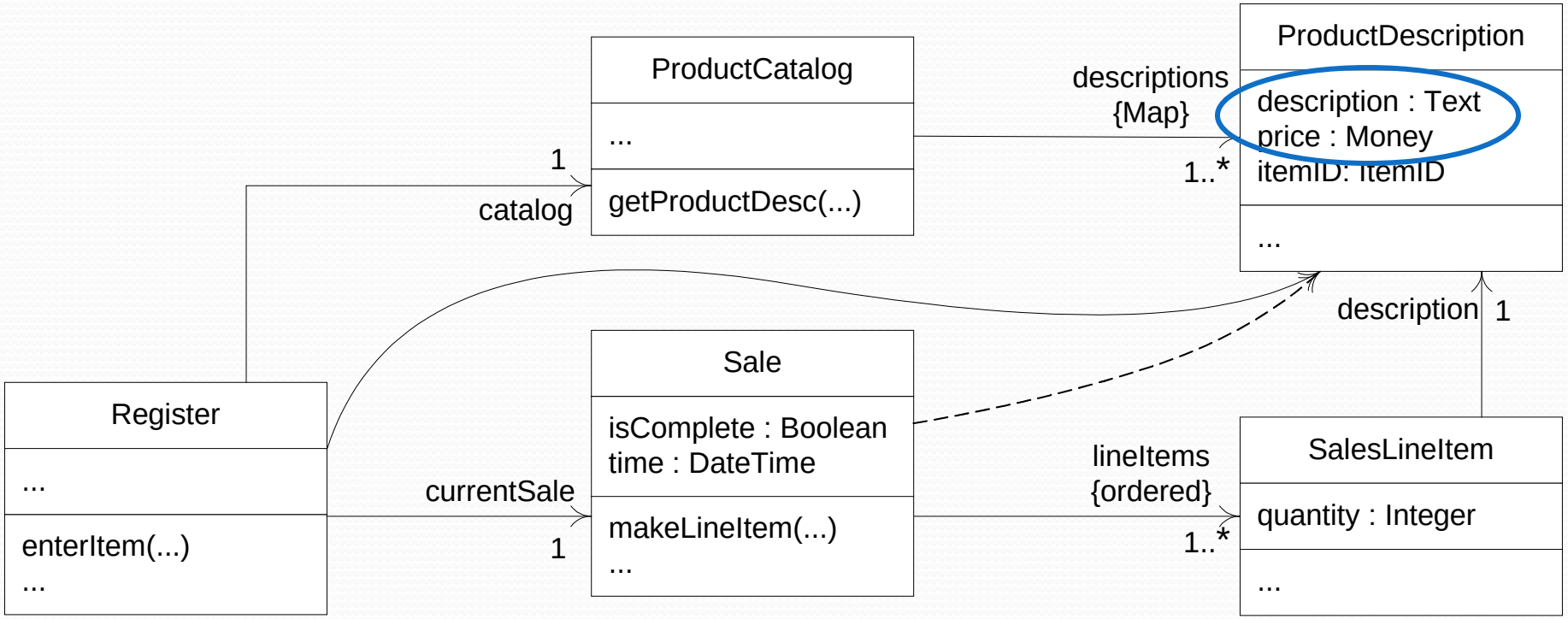
ProductDescription & Price Known?



- The information is in the ProductDesription class

Controller / Creators – enterItem

- Partial DCD – static view of enterItem



Larman fig. 18.8 Partial DCD static view of enterItem

Contract C03 – endSale

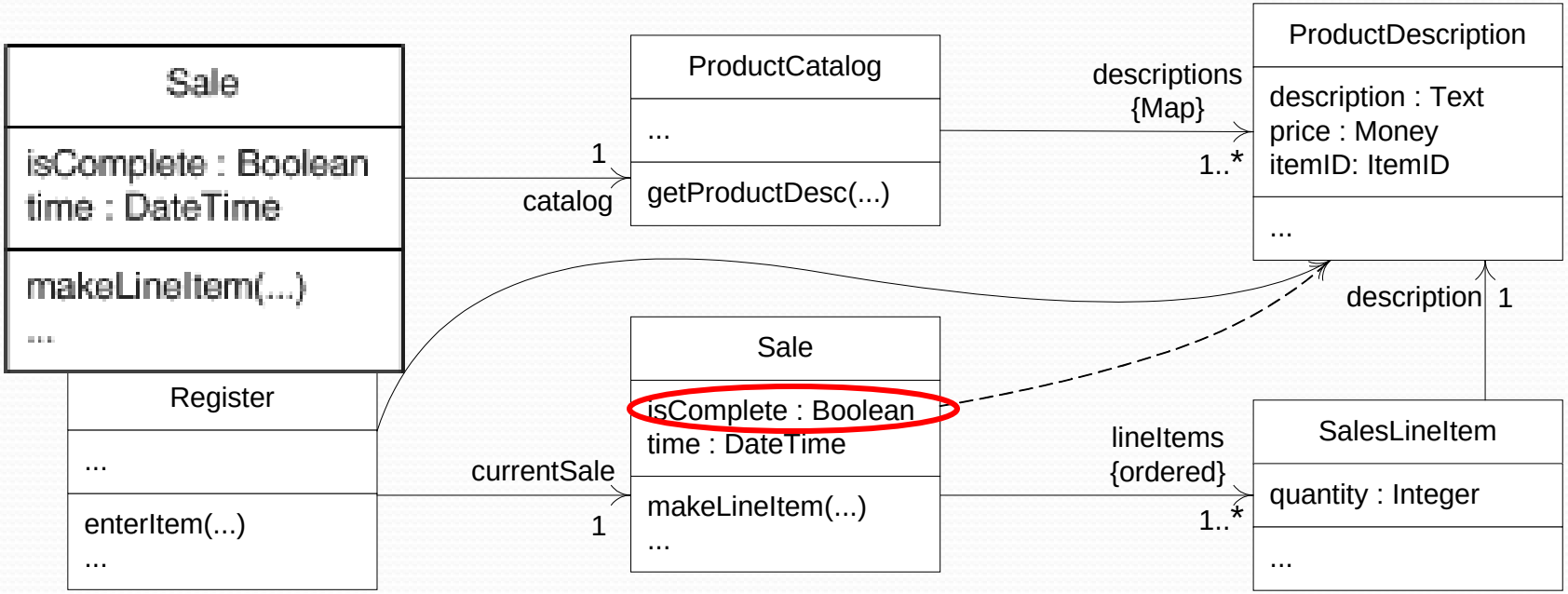
Operation:	endSale()
Cross References:	Use Case: Process Sales
Preconditions:	A sale is underway

Postconditions:

- Sale.isComplete became true (attribute modification)

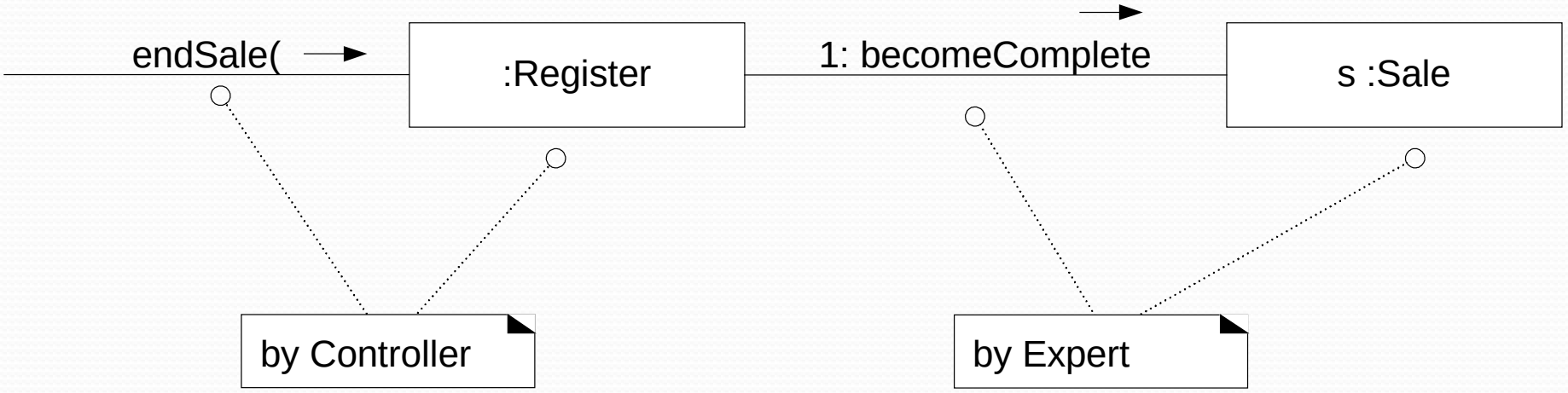
A few design decisions - endSale

- Information Expert should be the first design pattern to consider.
- Who should be responsible for setting the isComplete attribute of the Sale to true?



Controller / Expert – endSale

- :Register continues to be the controller for the system operation message of endSale. It sends the message to Sale to end the sale and s:Sale will set the isComplete attribute to true.



Larman fig. 18.19

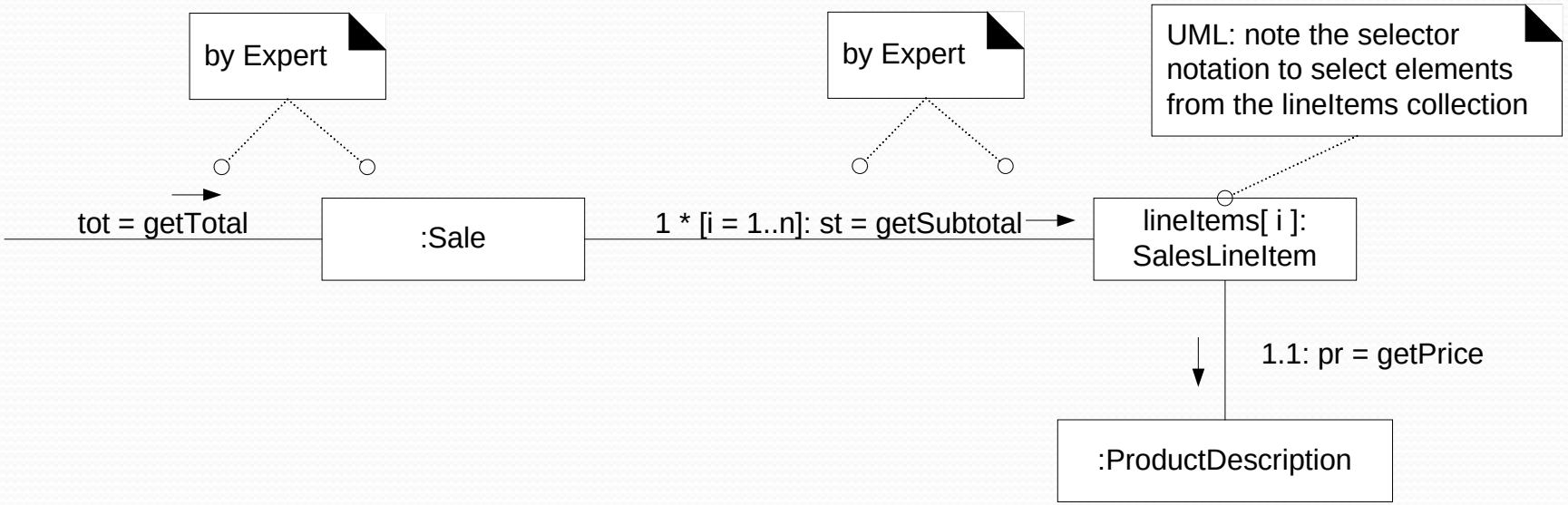
Need running Sale Total

1. Who should be responsible for calculating the Sale Total?
2. Who should be responsible for calculating the SaleLineItem subtotal?
3. Who should be responsible for providing the ProductDescription price?

	Information required for Sale Total	Knowing Responsibility	Implementation Method
1	All SalesLineItems in current Sale	Sale	getTotal
2	SalesLineItem.quantity	SaleLineItem	getSubtotal
3	ProductDescription.price	ProductDescription	getPrice

Expert – getTotal

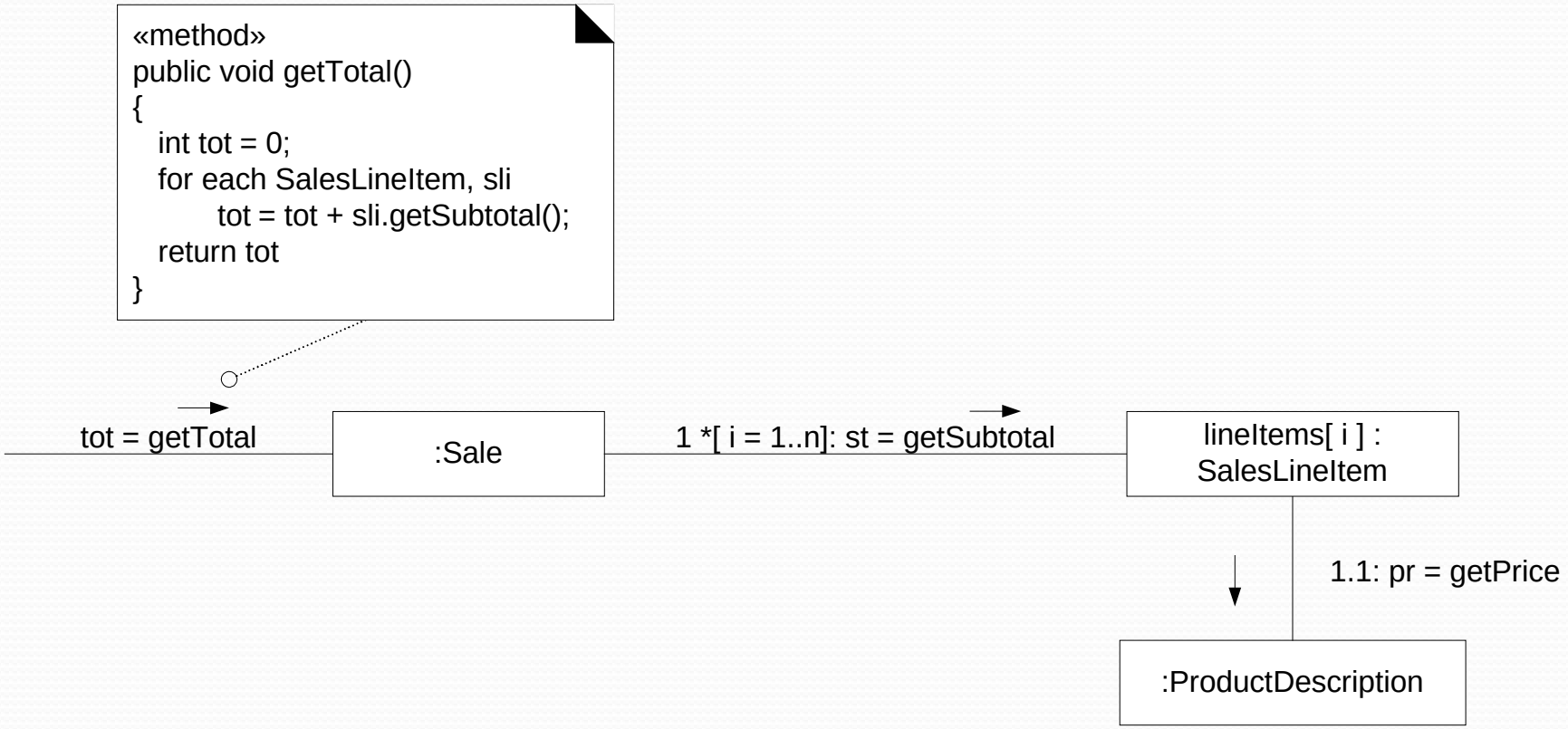
- Sale is responsible for knowing its total and requires ProductDescription, SalesLineITem, Sale



Larman fig. 18.10

getTotal Method

- The getTotal message most liekly will be an object in the UI layer such as JFrame



Larman fig. 18.11

Contract C04 – makePayment

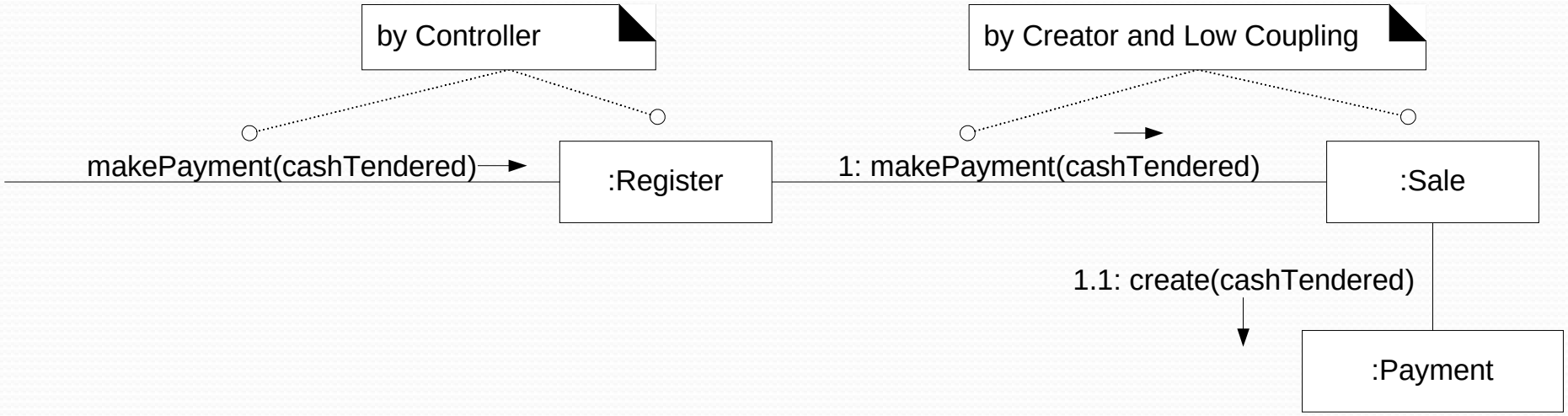
Operation:	makePayment()
Cross References:	Use Case: Process Sales
Preconditions:	A sale is underway

Postconditions:

- A Payment instance p was created (instance creation).
- p.amountTendered became amount (attribute modification).
- p was associated with current Sale (association formed).
- The current Sale was associated with the Store (association formed); (to add it to the historical log of completed sales).

Controller / Creator – makePayment

- Sales create the Payment for better cohesion and low coupling in the Register, thus lighten the Register responsibility



Larman fig. 18.13

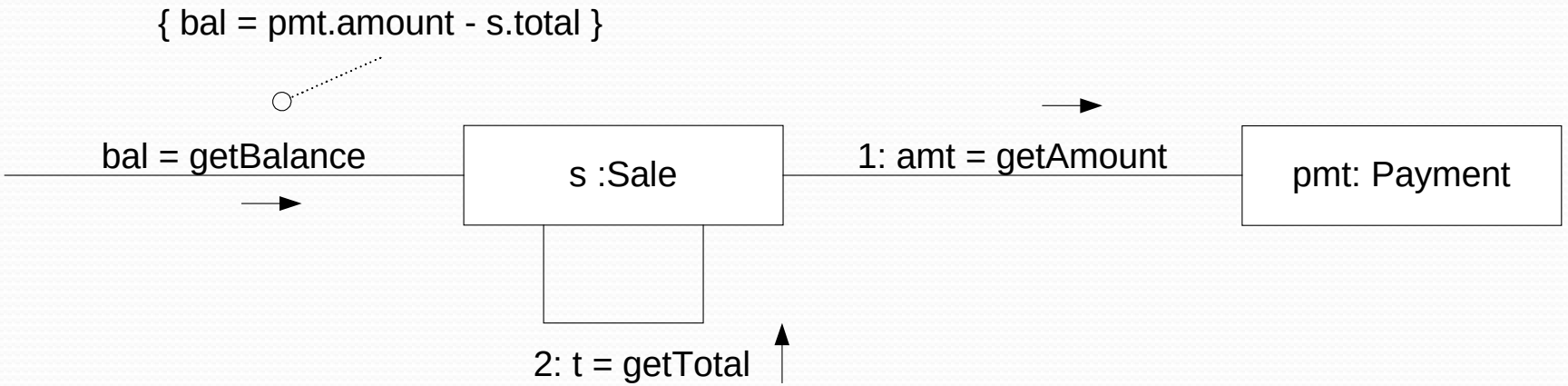
A few design decisions – Balance

- Who should be responsible for knowing the balance?
  Balance = salesTotal - amountTendered

1. Since Sale already has visibility to the amountTendered, as its creator, it is better design to let Sale, rather than Payment, to own the knowing responsibility

Expert – makePayment

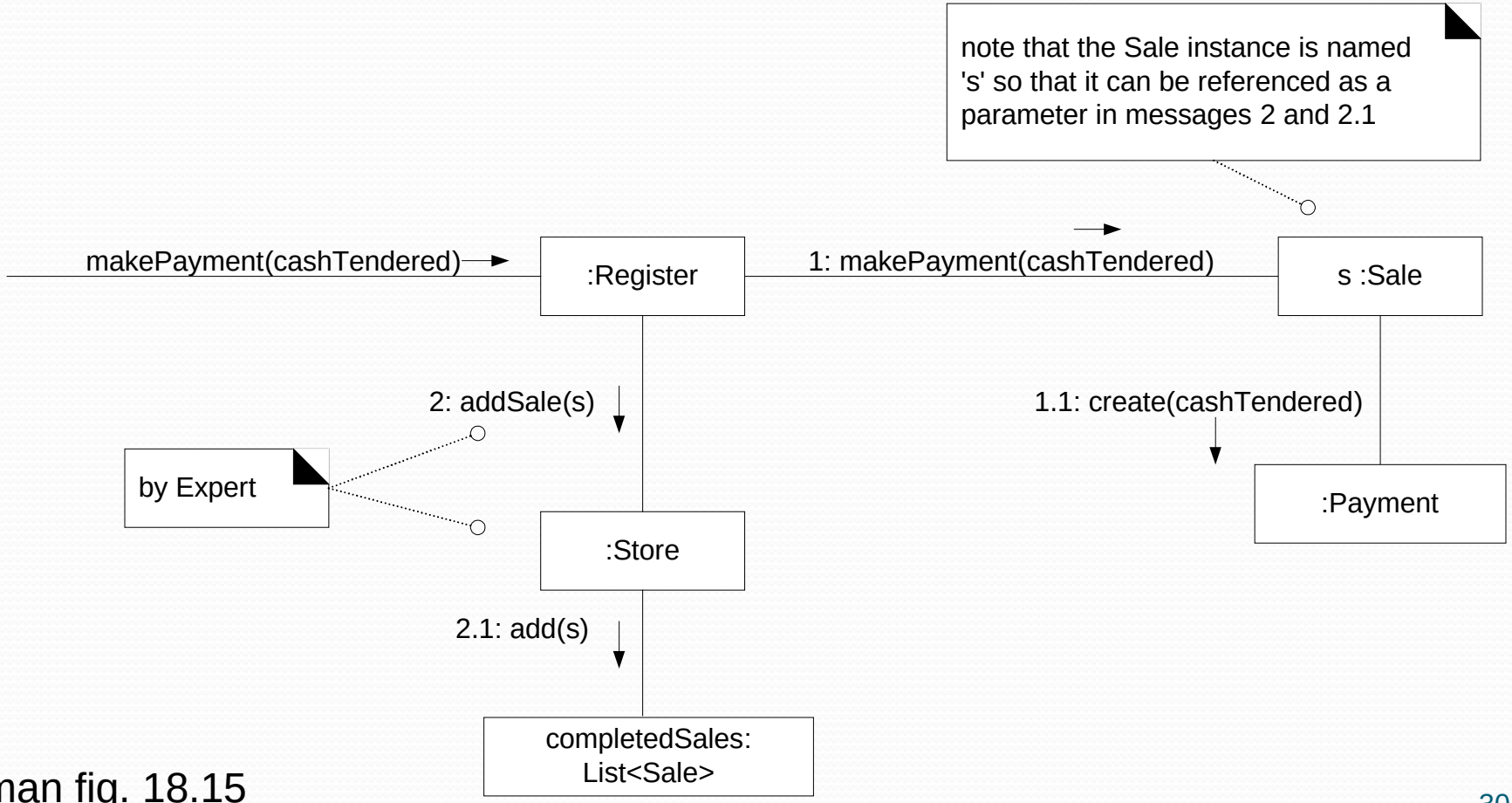
- Sale knows the balance (i.e. the sales total and payment tendered) from its visibility to Payment



Larman fig. 18.16

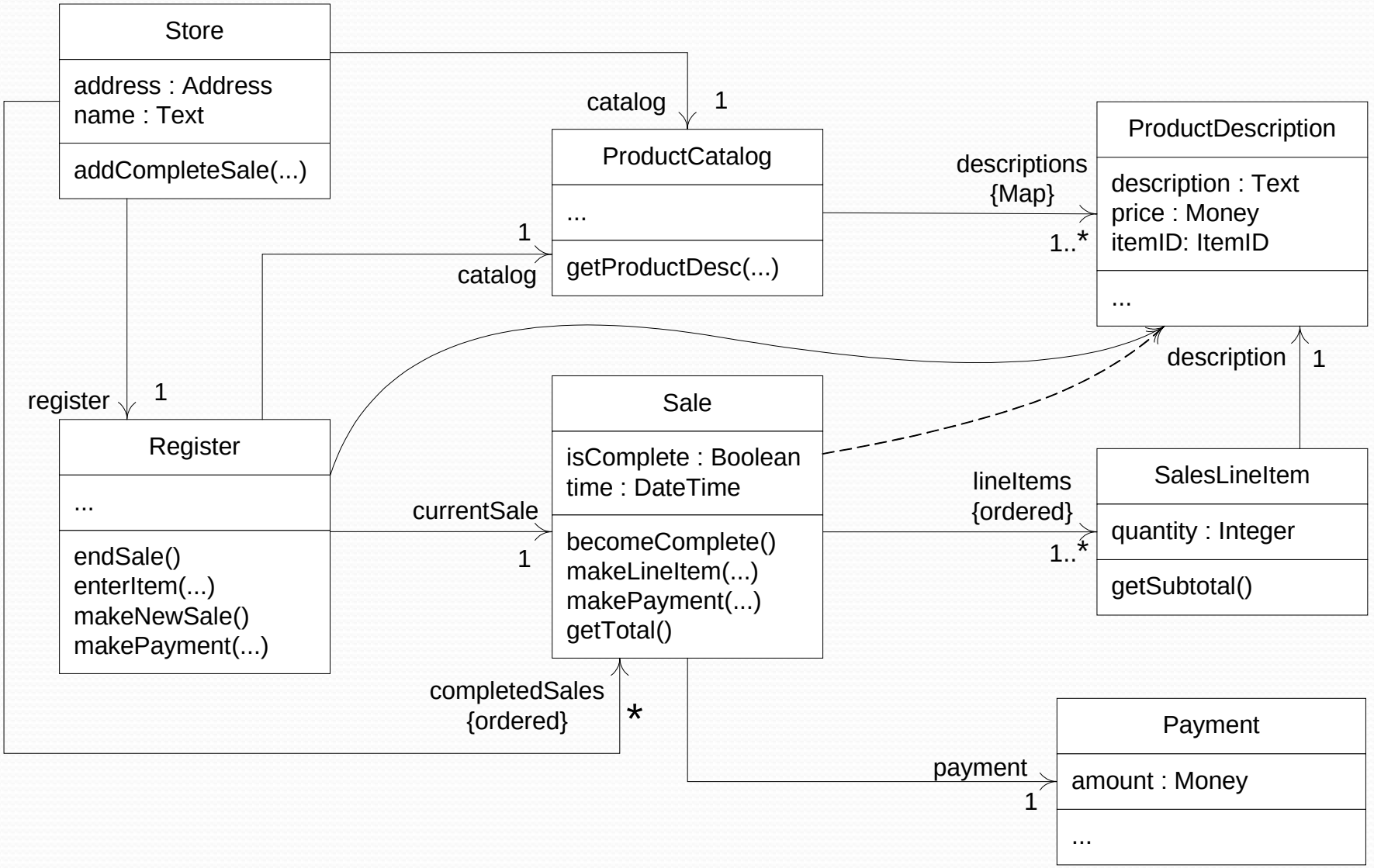
Expert – Log Completed Sale

- Use Store to keep list of completedSales per Operation Contract C04 postconditions



Larman fig. 18.15

NextGen POS – Updated DCD



Larman fig. 18.17