UNICORN
TAK SE DĚLÁ SOFTWARE

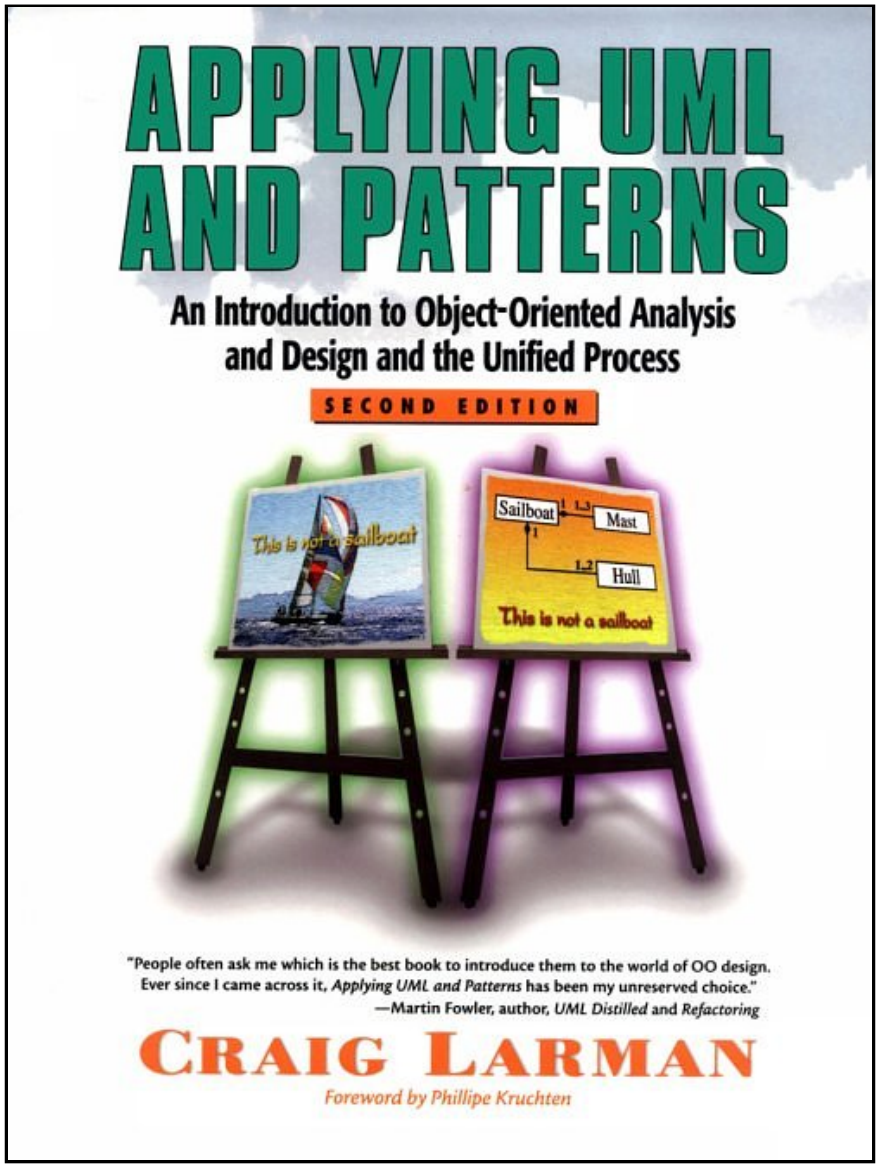
An Introduction to OOAD

Jiří Svačina, Unicorn a.s.

Based upon the book „Applying UML and Patterns“

written by Craig Larman

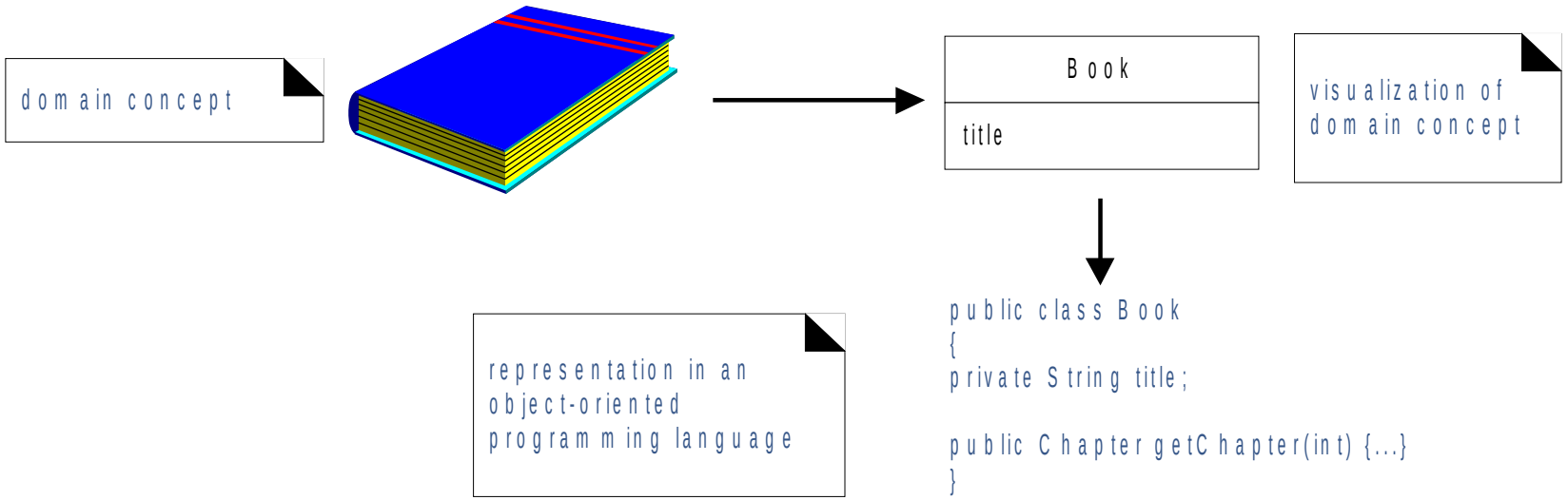
BASED ON THE BOOK...



WHAT IS ANALYSIS & DESIGN

- Analysis
  - Investigation of the problem and requirements rather than a solution
  - Requirements analysis
  - Object (domain) analysis
- Design
  - Conceptual solution that fulfills the requirements
  - Design can be implemented directly
  - Object design, database design...

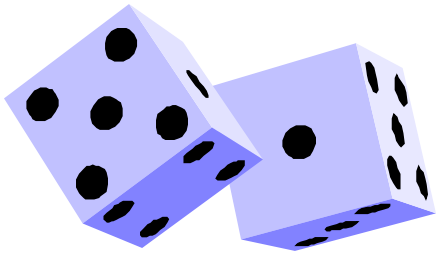
OBJECT-ORIENTED ANALYSIS & DESIGN



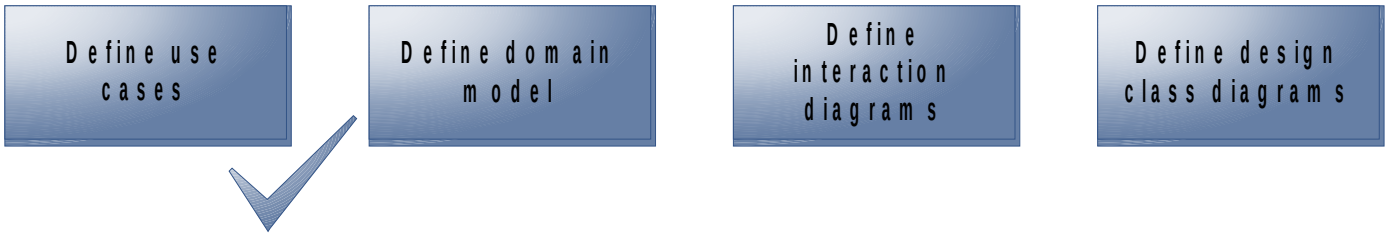
- Object-oriented analysis - emphasis on finding and describing objects (concepts) in the problem domain
- Object-oriented design - emphasis on defining software objects and how they collaborate to fulfill the requirements

EXAMPLE - DICE GAME

- Player rolls two die
- If the total is seven, the player wins, otherwise the player loses

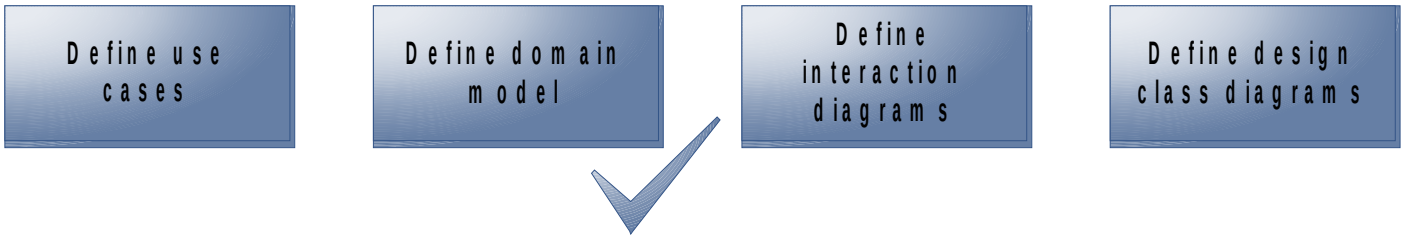


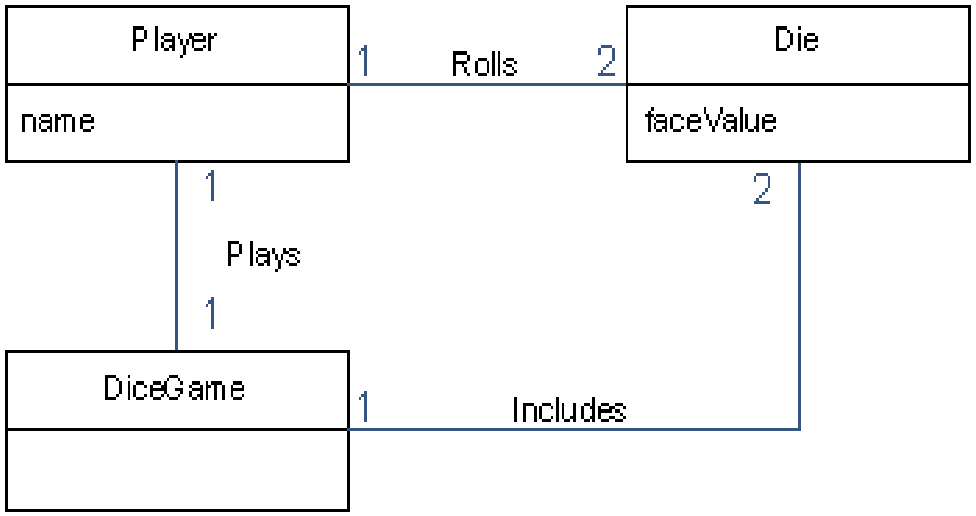
EXAMPLE - DEFINE USE CASES



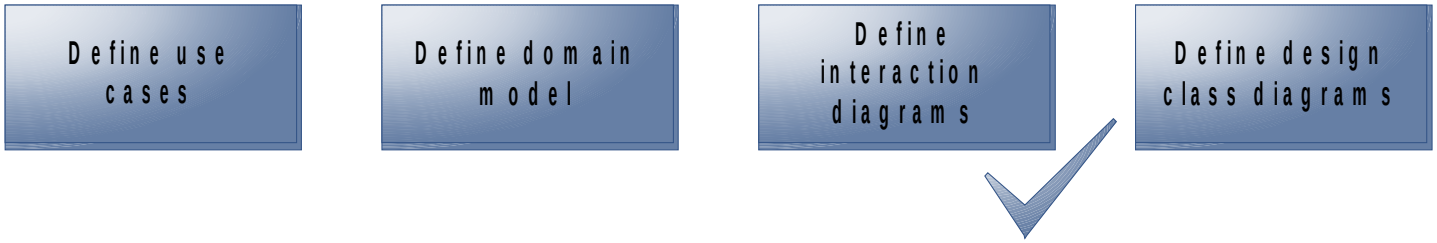
- „Play a dice game“ use case brief description
  - A player picks up and rolls the dice. If the dice face value totals seven, he wins; otherwise, he loses.

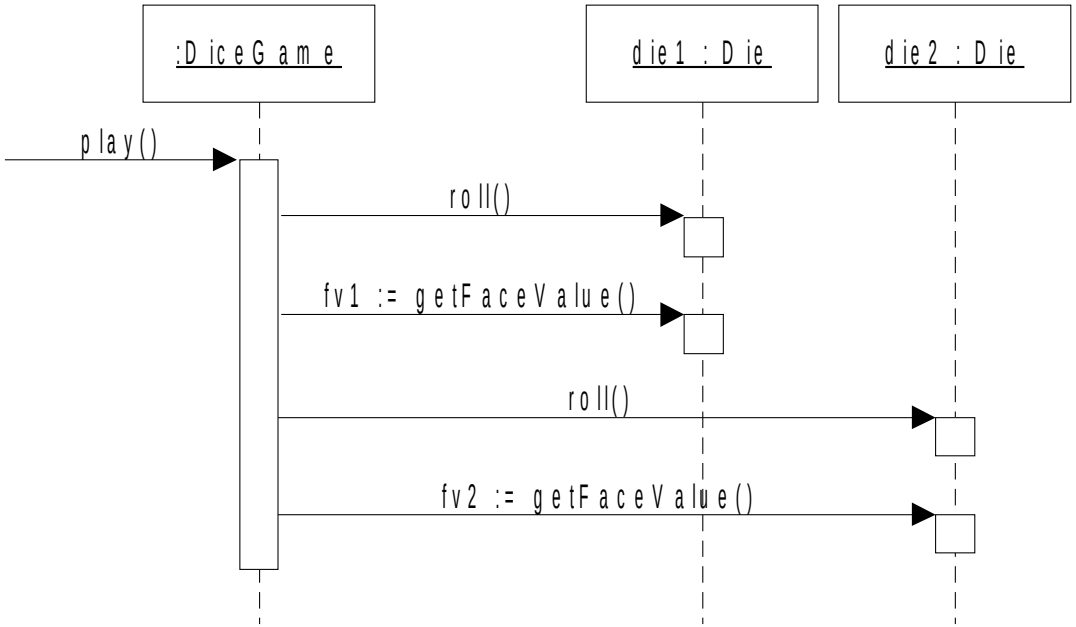
EXAMPLE - DOMAIN MODEL



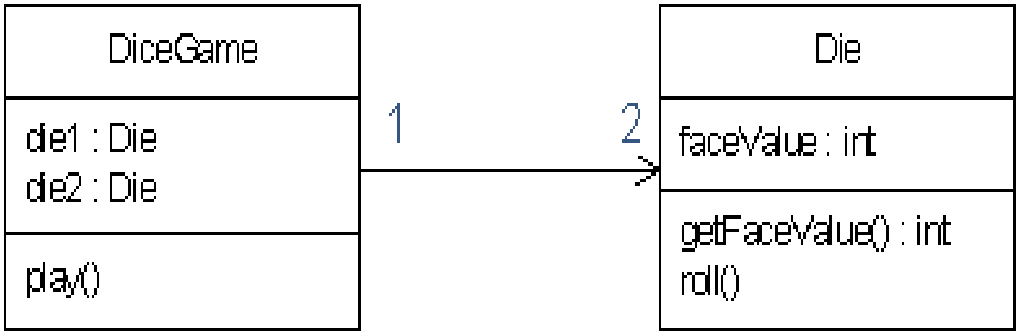


EXAMPLE - DEFINE INTERACTION DIAGRAMS





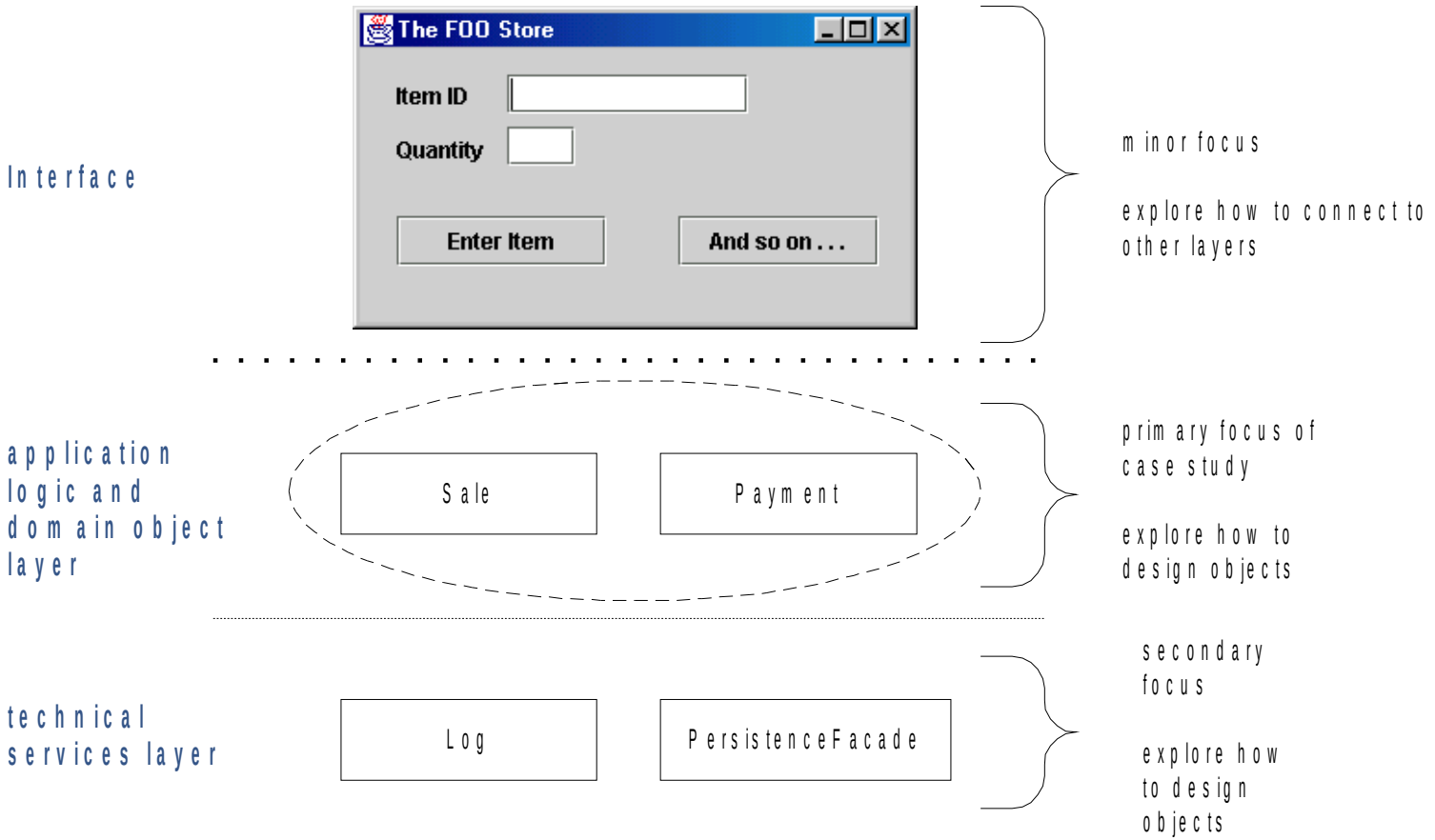
EXAMPLE - DESIGN CLASS DIAGRAMS



CASE STUDY - POS SYSTEM

- Point-of-sale system
- Records sales and handles payments (retail stores)
- Computer, bar code scanner, etc.
- Third party applications (tax calculator, inventory control)
- Fault tolerance
- Various output devices
- Flexibility and customization

CASE STUDY - ARCHITECTURAL LAYERS



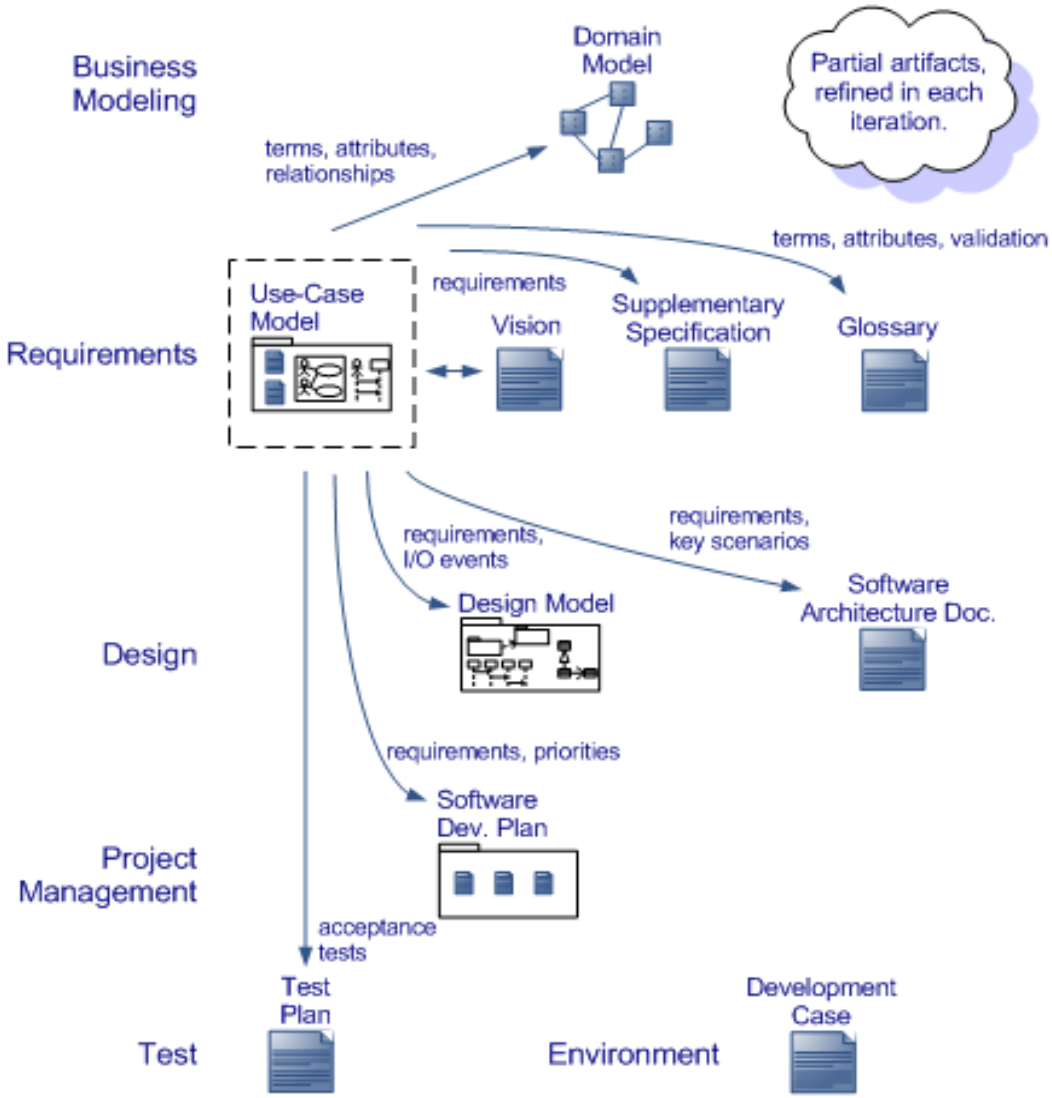
USE CASES

- A technique for requirements management
- Functional requirements

- Goal (need) of a customer or end user

- Example - Process sale
  A customer arrives at the checkout with items to purchase. The cashier uses the POS system to record each purchased item. The system presents a running total and line-item details. The customer enters payment information, which the system validates and records. The system updates inventory. The customer receives a receipt from the system and then leaves with the items.

USE CASES IN CONTEXT



SYSTEM SEQUENCE DIAGRAMS

Simple cash-only *Process Sale* scenario:

1. Customer arrives at a POS checkout with goods and/or services to purchase.
2. Cashier starts a new sale.

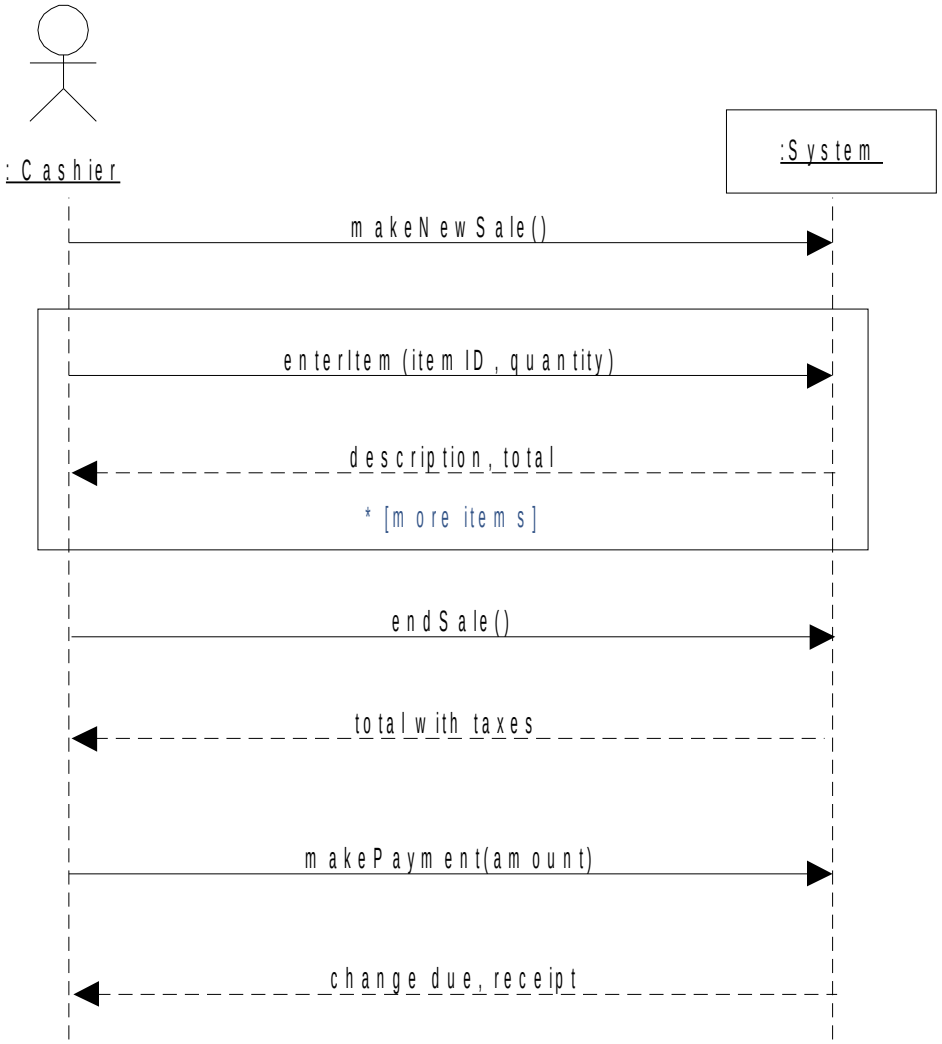
3. Cashier enters item identifier.
4. System records sale line item and presents item description, price, and running total.

Cashier repeats steps 3-4 until indicates done.

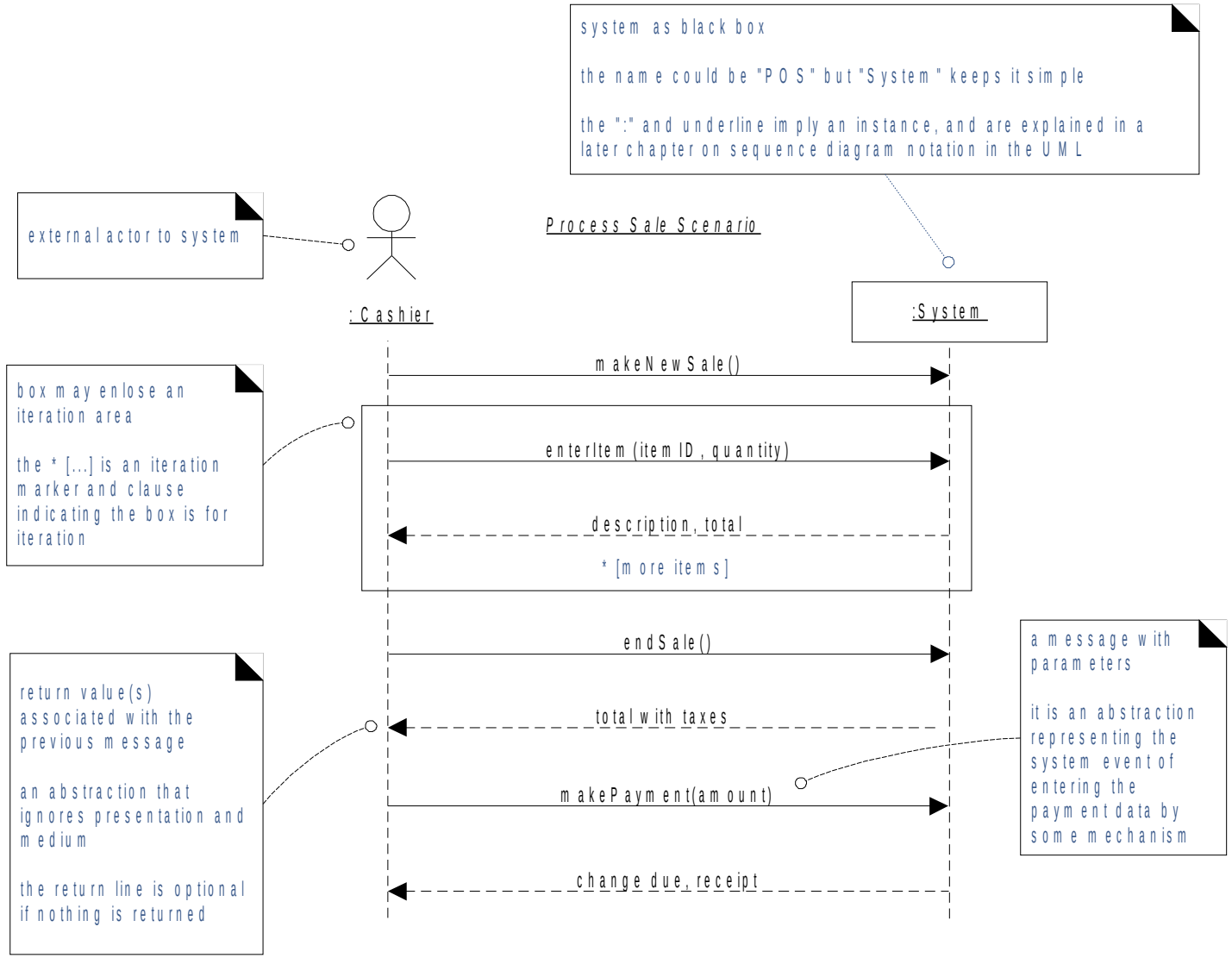
5. System presents total with taxes calculated.

6. Cashier tells Customer the total, and asks for payment.
7. Customer pays and System handles payment.

...



SYSTEM SEQUENCE DIAGRAMS



DOMAIN MODEL

- A domain model is a representation of real-world conceptual classes, not of software components. It is not a set of diagrams describing software classes or software objects with responsibilities

- Domain objects or conceptual classes
- Associations, aggregations, compositions or generalizations between conceptual classes
- Attributes of conceptual classes
- Other business rules (constraints)

NOUN PHRASE IDENTIFICATION

- Identify the nouns and noun phrases in textual descriptions of a domain (for example use cases)

- Main Success Scenario (or Basic Flow):

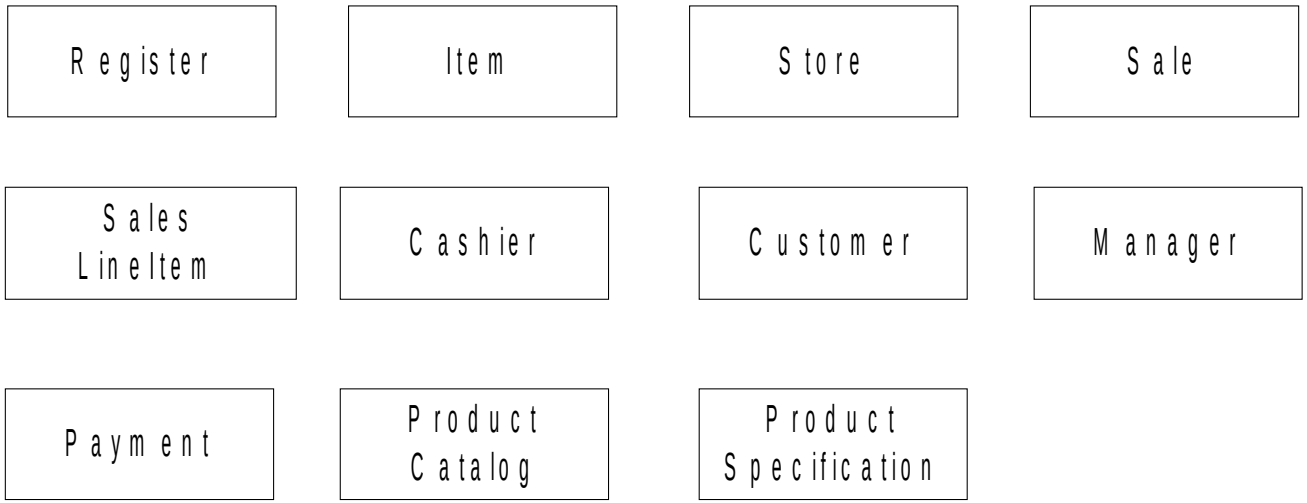
1. Customer arrives at POS checkout with goods and/or services to purchase.
2. Cashier starts a new sale.
3. Cashier enters item identifier.
4. System records sale line item and presents item description, price, and running total. Price calculated from a set of price rules.

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.
8. System logs completed sale and sends sale and payment information to the external Accounting system (for accounting and commissions) and Inventory system (to update inventory).
9. System presents receipt.
10. Customer leaves with receipt and goods (if any).

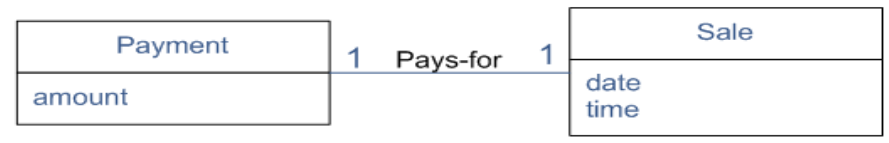
...

CANDIDATE CLASSES FOR SALES DOMAIN



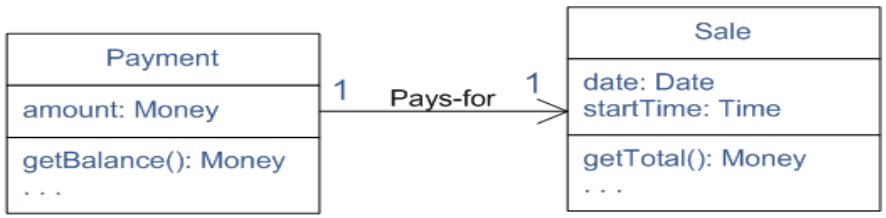
- According to our current understanding of the problem
- Related to Glossary

MODELS - MULTIPLE PERSPECTIVES



UP Domain Model

Raw UML class diagram notation used in an essential model visualizing real-world concepts.



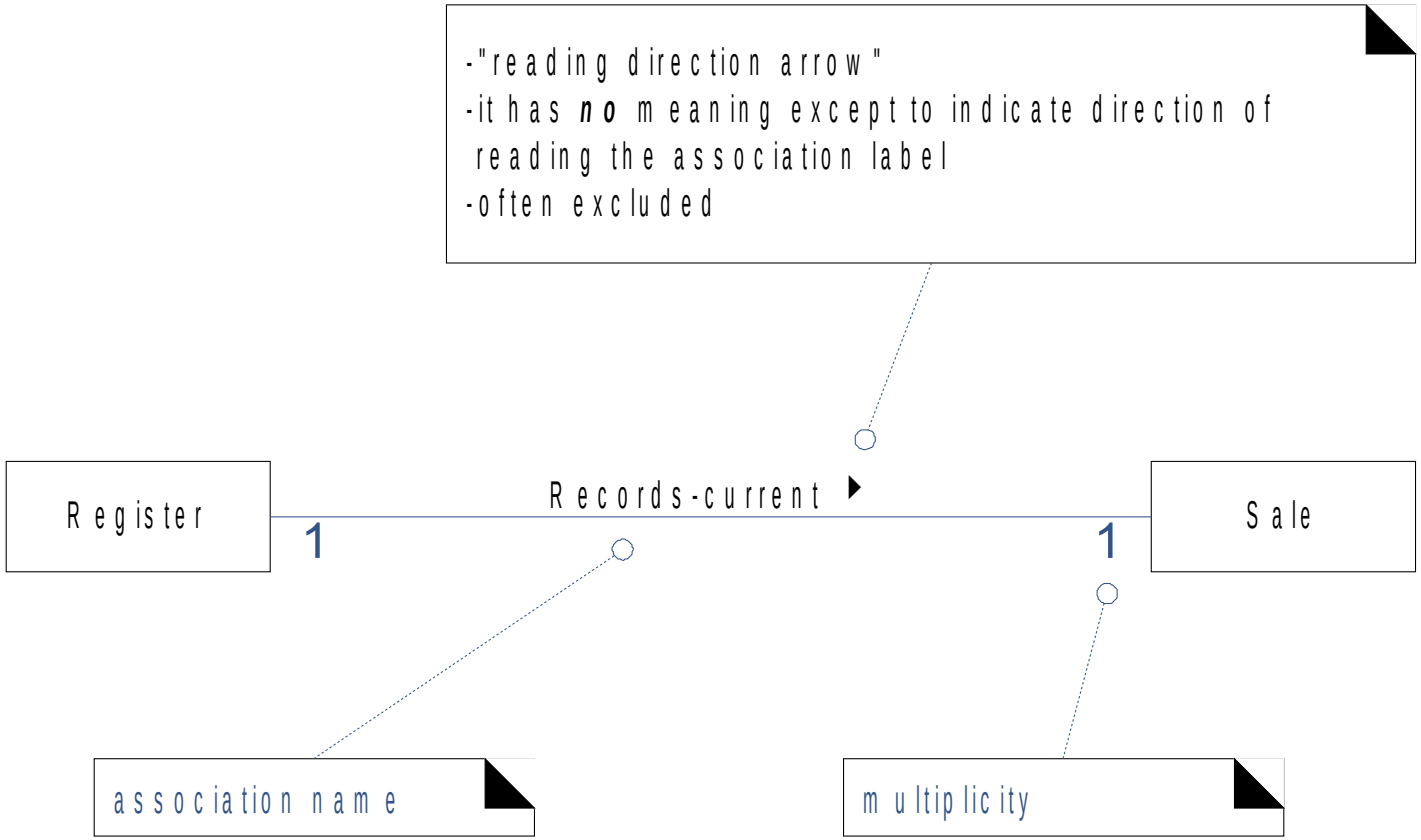
UP Design Model

Raw UML class diagram notation used in a specification model visualizing software components.

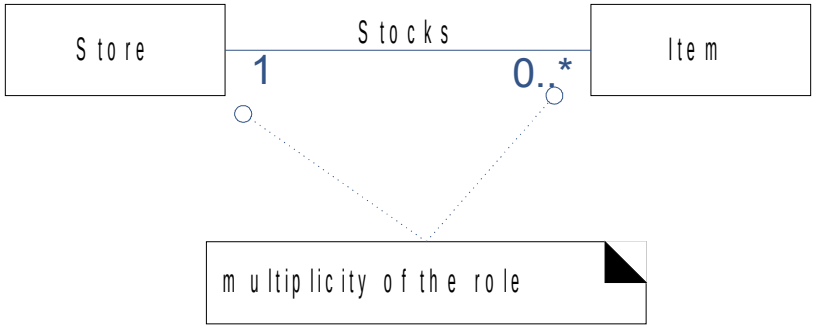
- Conceptual perspective - requirements
  - Diagrams interpreted as describing things in the real world
- Specification perspective - analysis
  - Diagrams interpreted as describing software components without implementation details
- Implementation perspective - design
  - Diagrams interpreted as describing actual implementation (code)

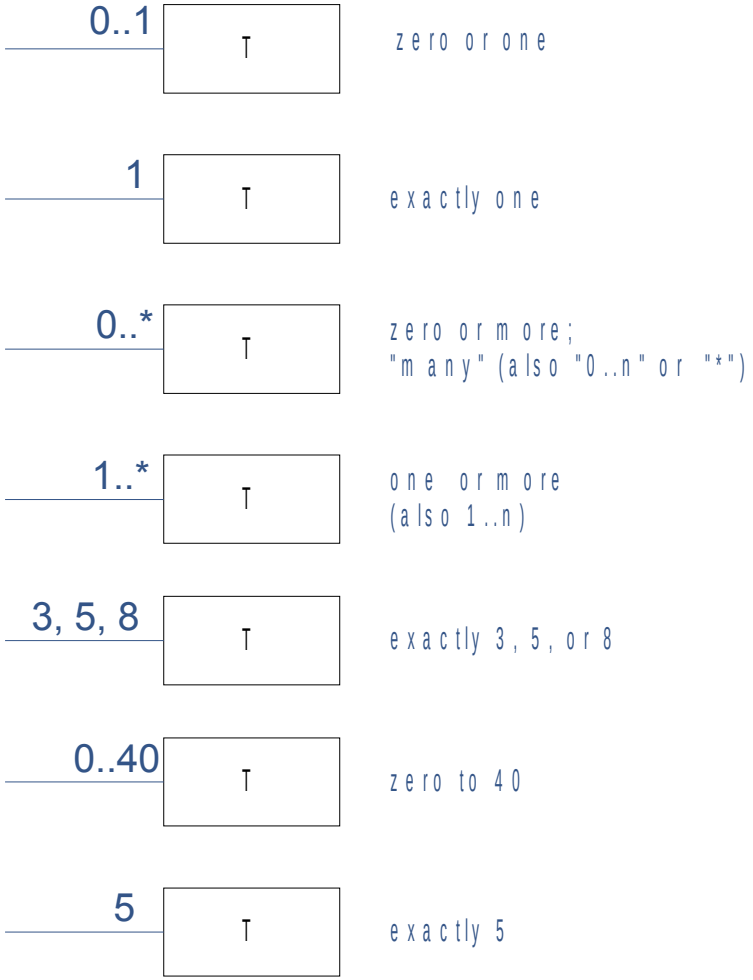
DOMAIN MODEL - ASSOCIATIONS

- The concepts do have relationships

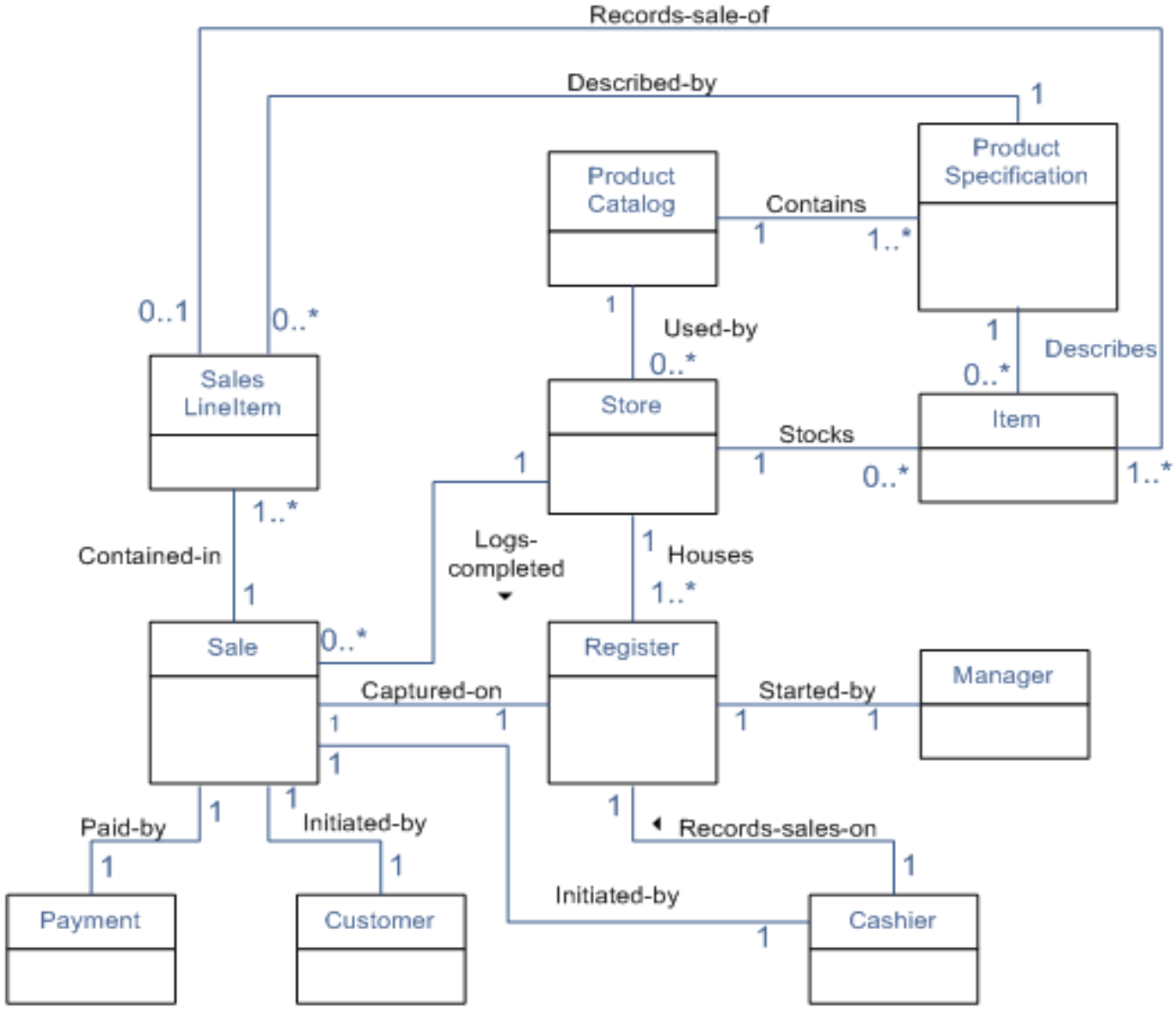


ASSOCIATION MULTIPLICITIES



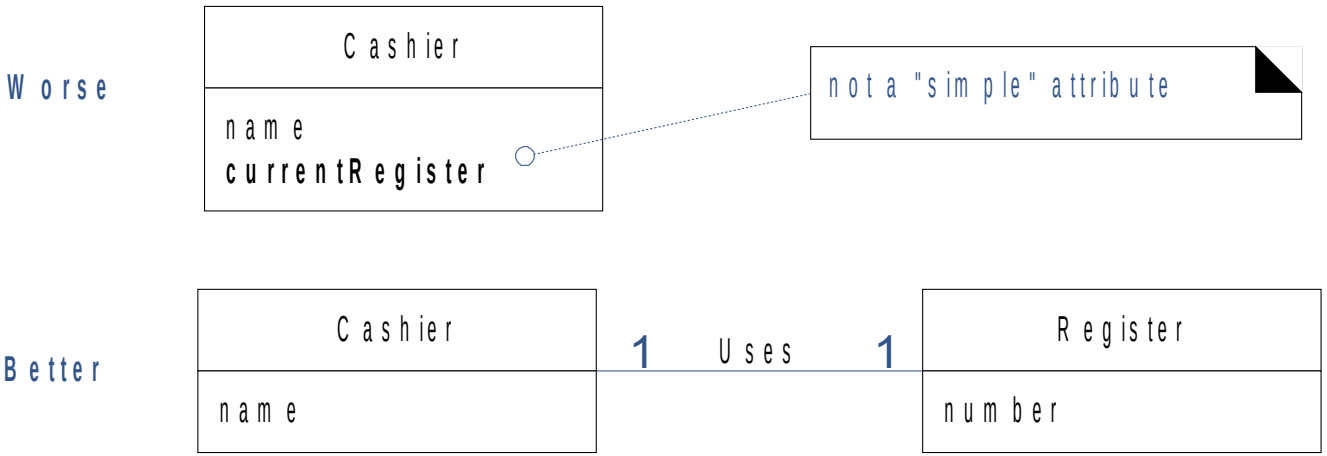


POS ASSOCIATIONS

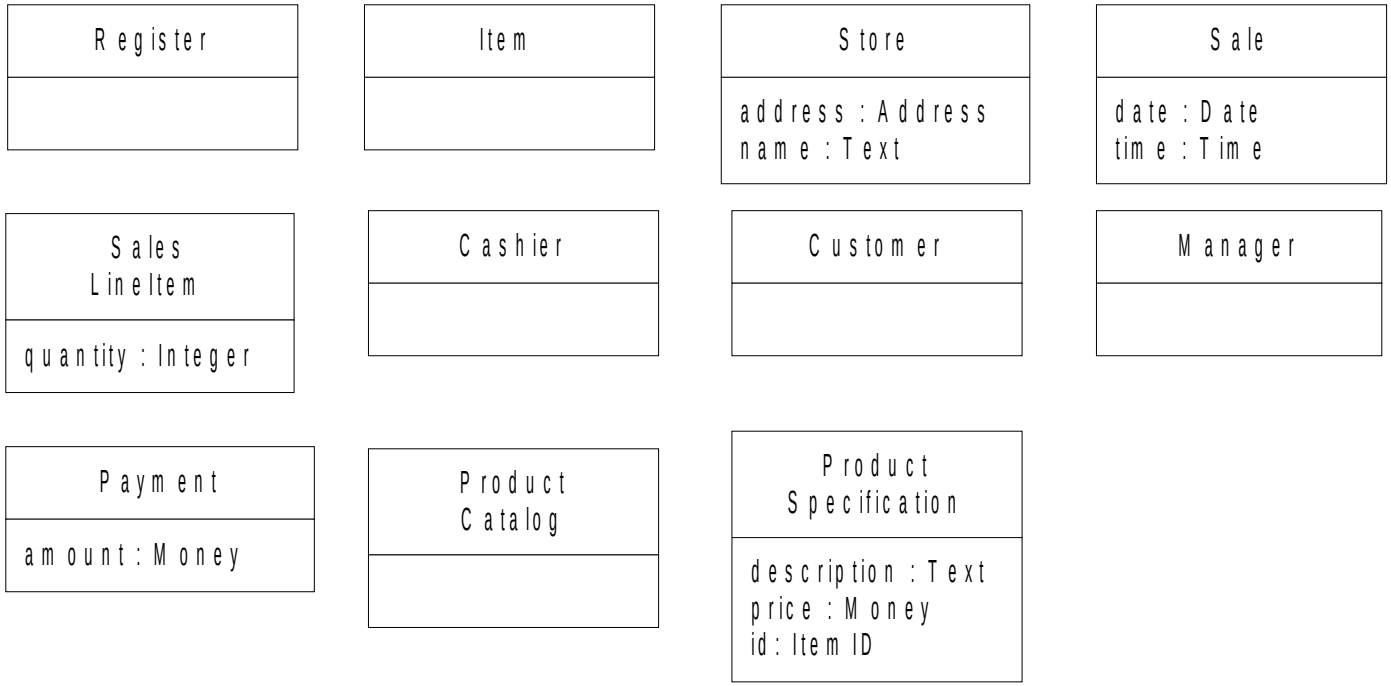


ATTRIBUTES

- In a domain model include attributes for which the requirements imply a need to remember information
- Keep attributes simple

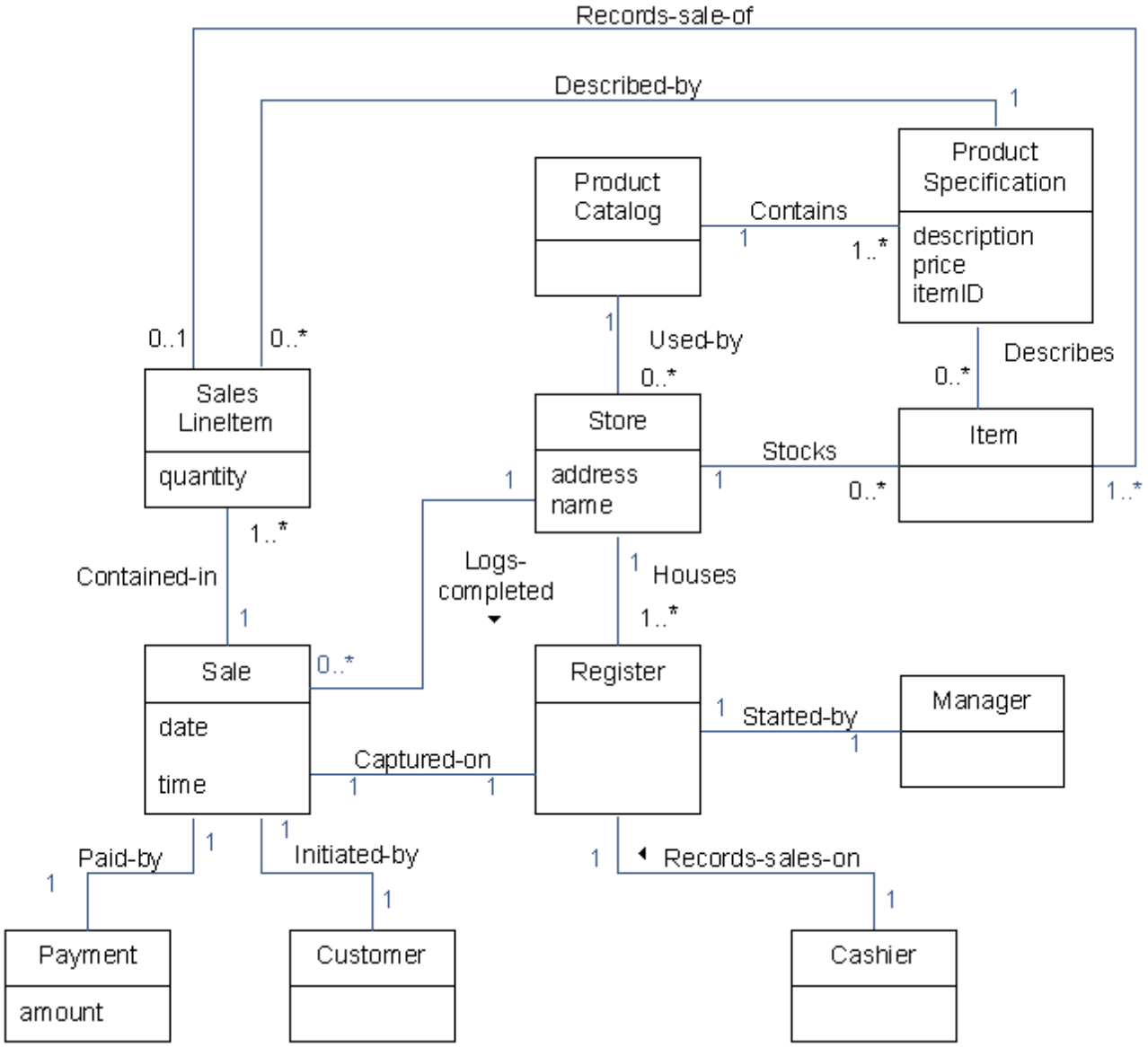


ATTRIBUTES IN POS DOMAIN MODEL



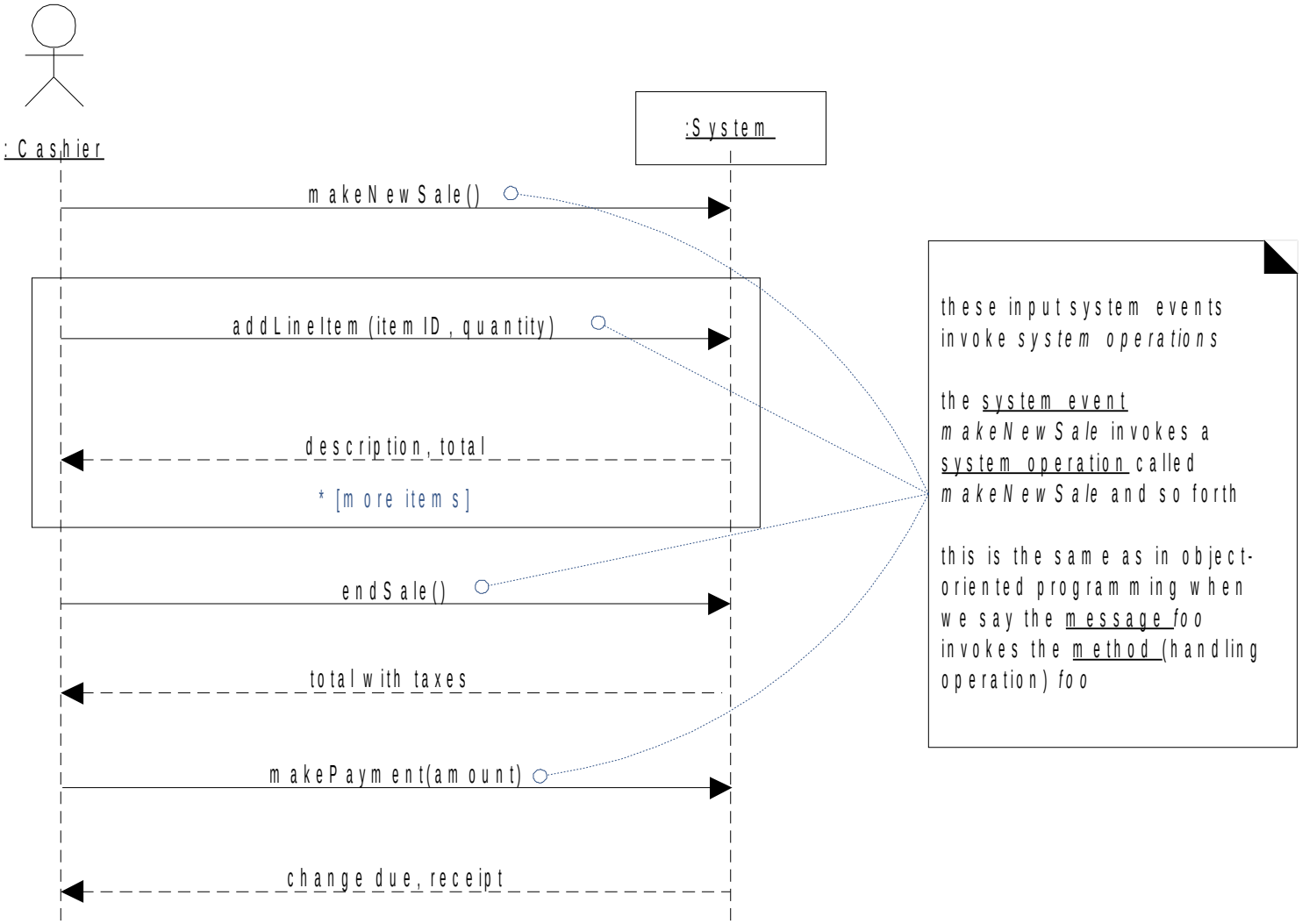
- According to our current understanding of the problem
- Does not have to be perfect at the moment

CONCLUSION - PARTIAL DOMAIN MODEL



OPERATION CONTRACTS

- Contracts can be defined for system operations



EXAMPLE CONTRACT - MAKENEWSALE

- *Operation*: makeNewSale()
- *Cross references*: Use Cases: Process Sale
- *Preconditions*: None
- *Postconditions:*
  - A Sale instance **s** was created (instance creation)
  - **s** was associated with the Register (association formed)
  - Attributes of **s** were initialized

EXAMPLE CONTRACT - 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)
  - **sli** was associated with current Sale (association formed)
  - **sli**.quantity became quantity (attribute modification)
  - **sli** was associated with a ProductSpecification, based on itemID match (association formed)

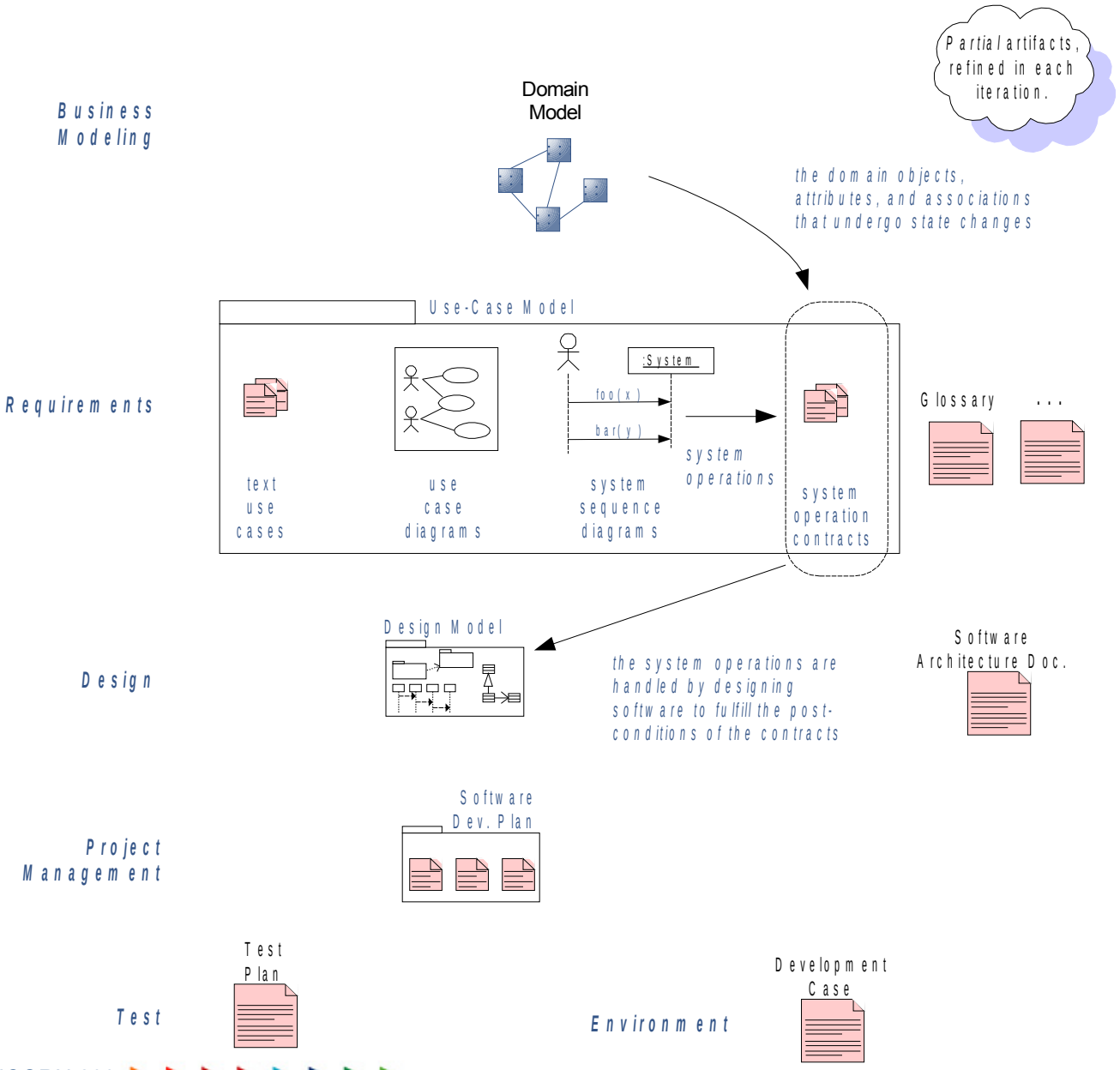
EXAMPLE CONTRACT - ENDSALE

- *Operation*:endSale()
- *Cross references*: Use Cases: Process Sale
- *Preconditions*: There is a sale underway
- *Postconditions*:
  - Sale.IsComplete becomes true
    (attribute modification)

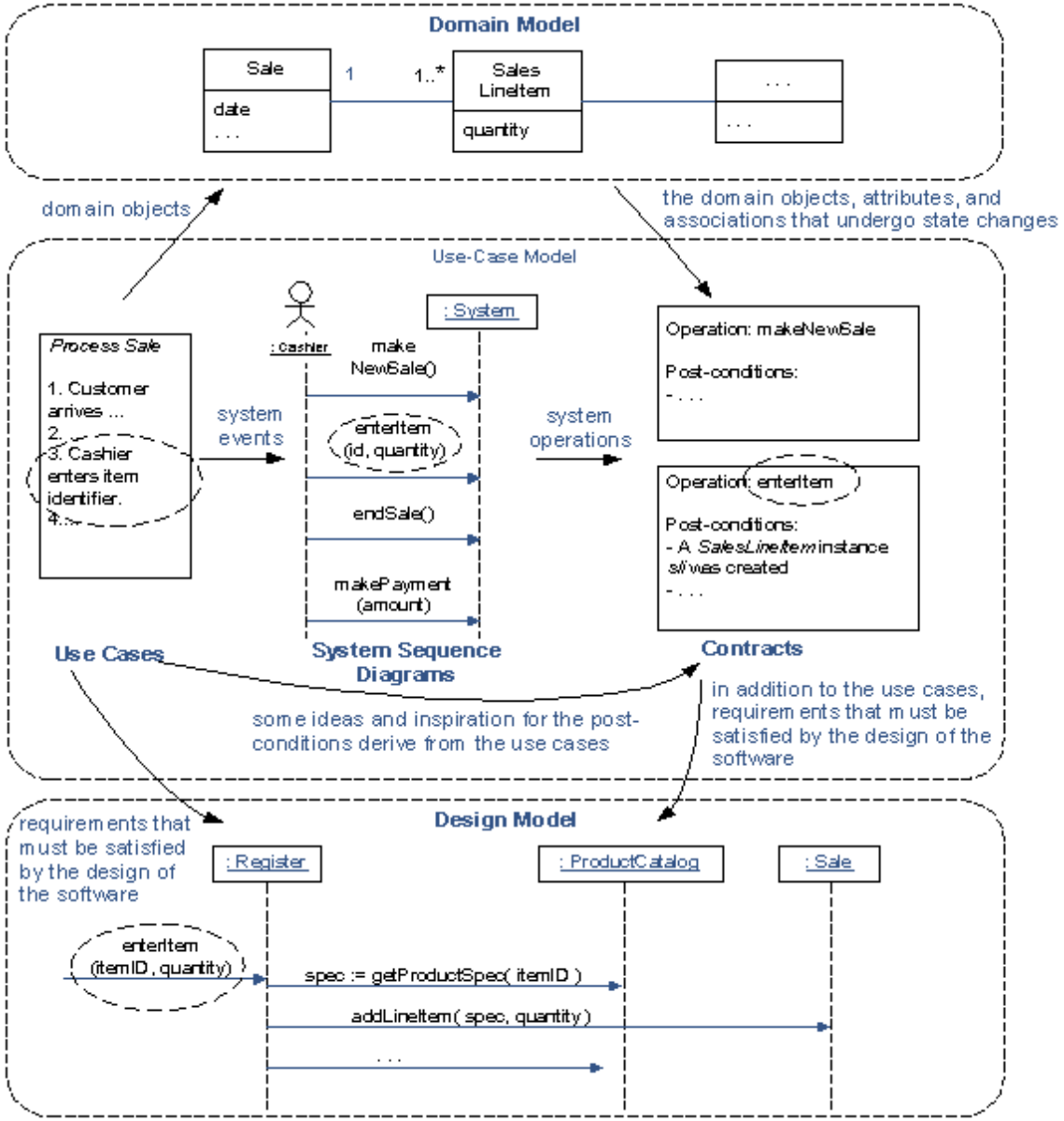
EXAMPLE CONTRACT - MAKEPAYMENT

- *Operation*: makePayment (amount : Money)
- *Cross references*: Use Cases: Process Sale
- *Preconditions*: There is a completed sale underway
- *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

METHODOLOGY - ARTIFACTS



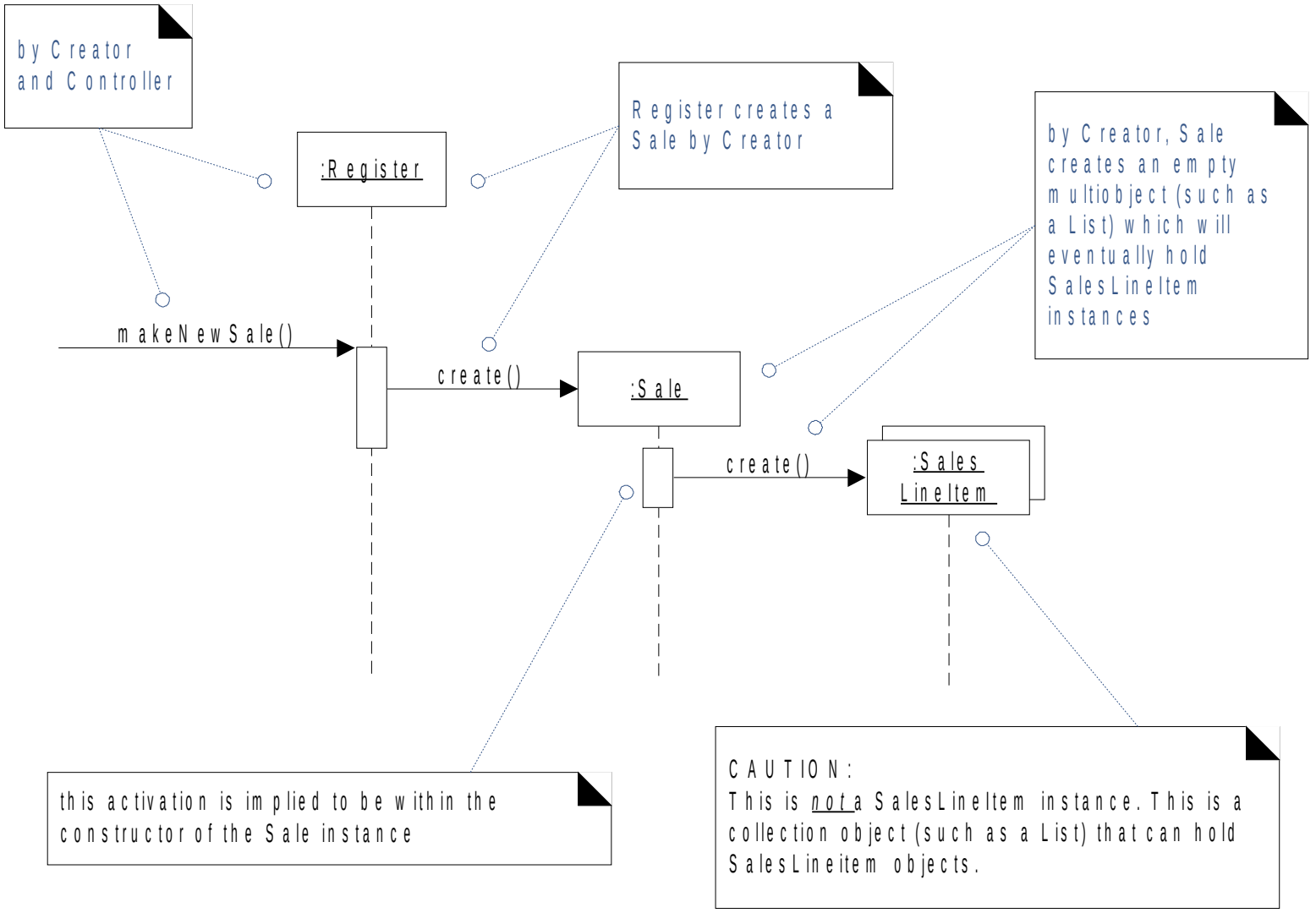
CONTRACT RELATIONSHIP TO OTHER ARTIFACTS



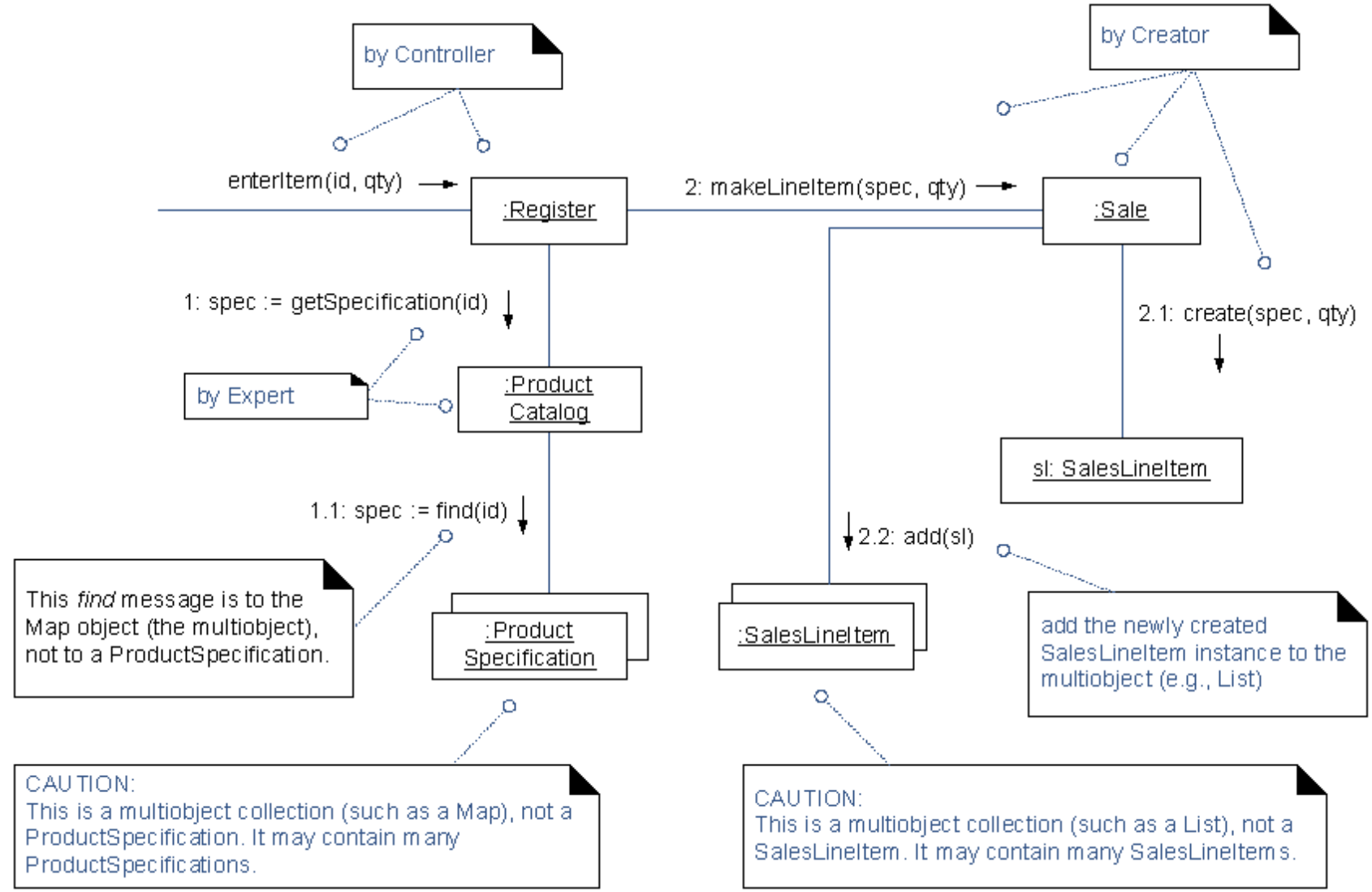
OBJECT RESPONSIBILITIES

- Responsibility - „a contract or obligation of classifier“
- **Doing** repsonsibilities of an object include
  - Doing something itself, such as creating an object or doing a calculation
  - Initiating action on other object(s)
  - Controlling and coordinating activitities in other objects
- **Knowing** responsibilities of an object include
  - Knowing about private encapsulated data
  - Knowing about related objects
  - Knowing about things it can derive or calculate
- Responsibilities are assigned to classes and later to methods

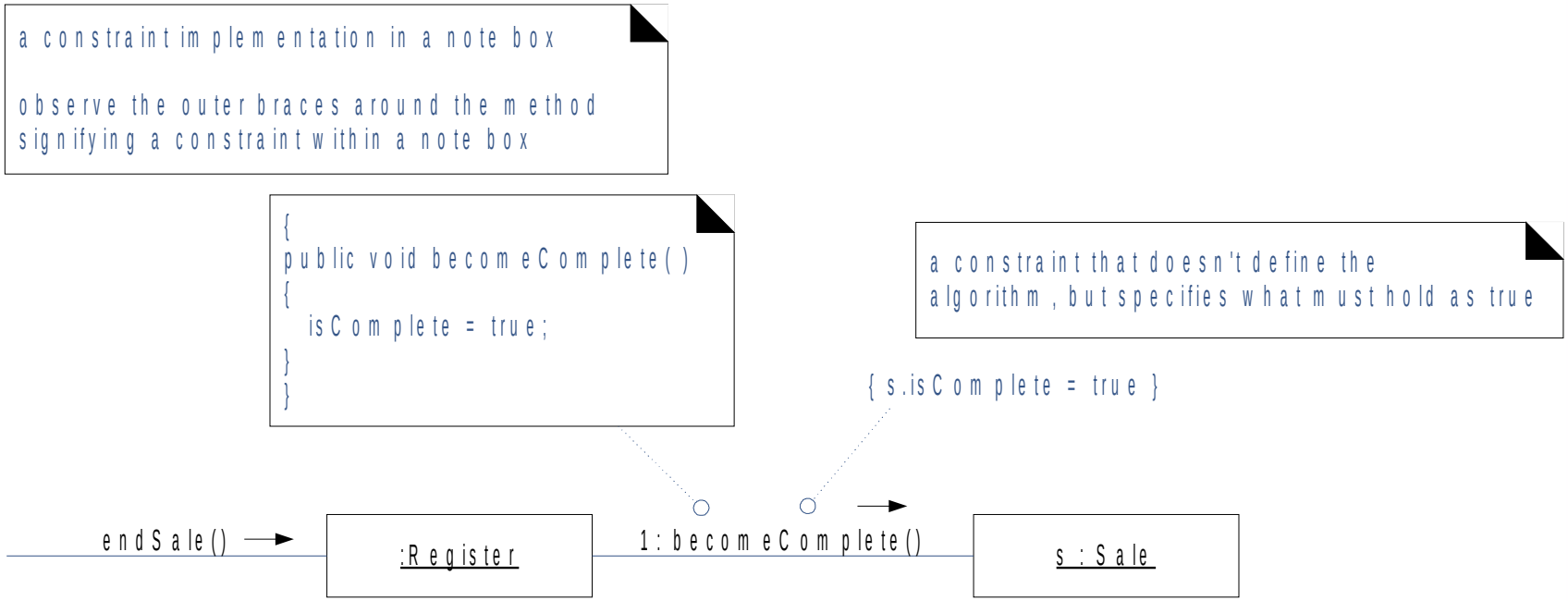
OBJECT DESIGN - makeNewSale



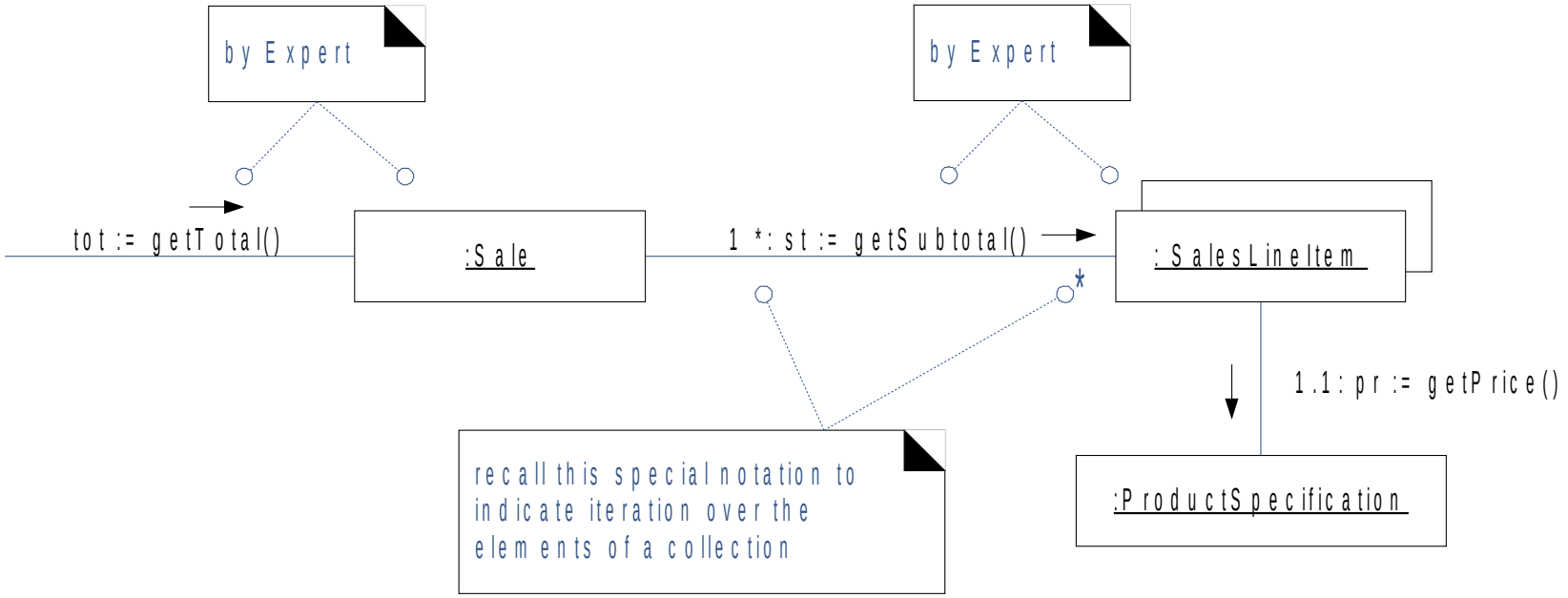
OBJECT DESIGN - enterItem



OBJECT DESIGN - endSale



OBJECT DESIGN - getTotal



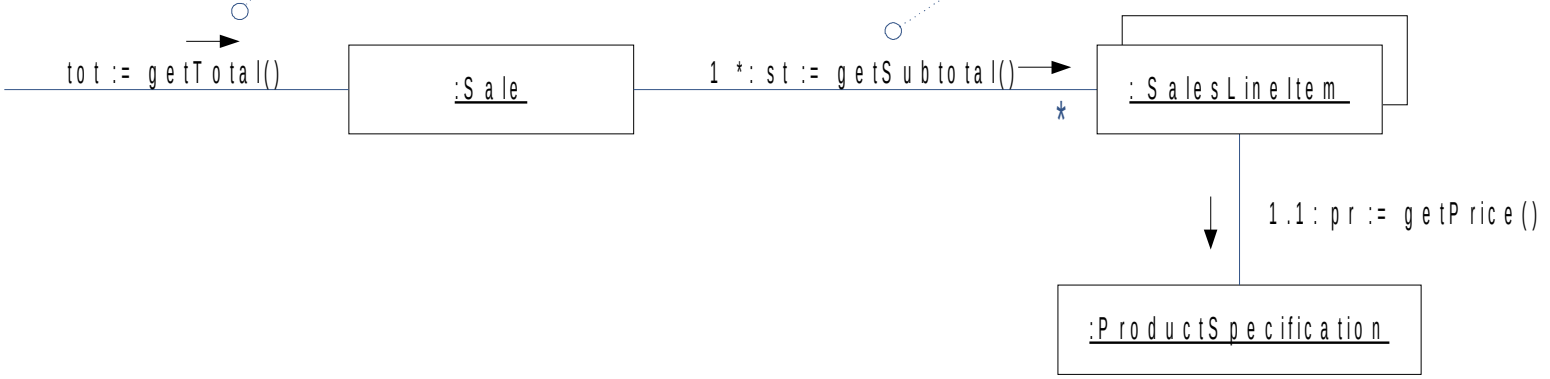
OBJECT DESIGN - getTotal

```
// observe the seudo code style here
{
public void getTotal()
{
   int tot = 0;
   for each SalesLineItem, sli
        tot = tot + sli.getSubtotal();
   return tot
}
}
```

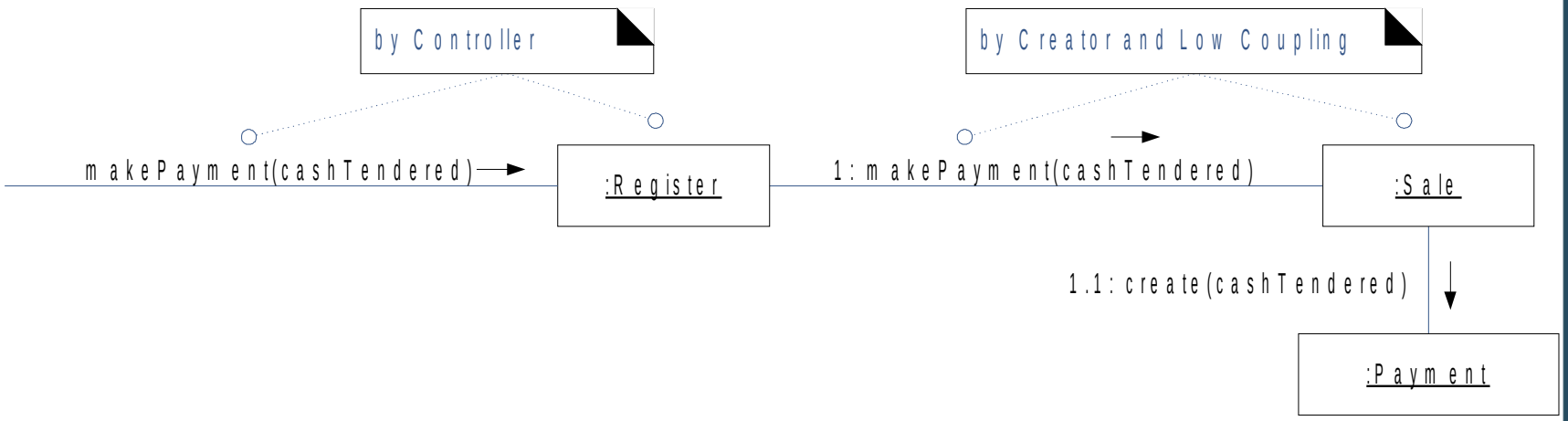
Note the semi-formal style of the constraint. "aSLI" is not formally defined, but most developers will reasonably understand this to mean an instance of SalesLineItem. Likewise with the expression aSLI.prodSpec.price.

The point is that the constraint language can be informal, to support quick and easy writing, if desired.

{ st = aSLI.quantity * aSLI.prodSpec.price }

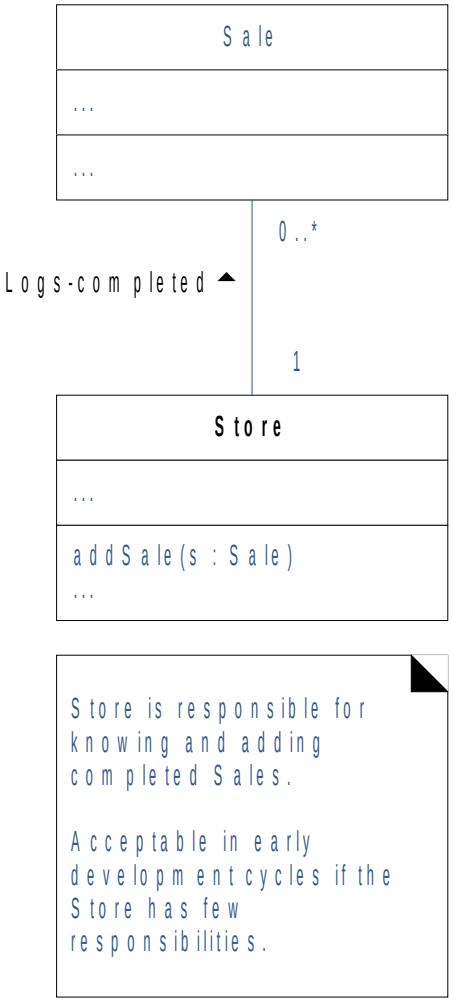


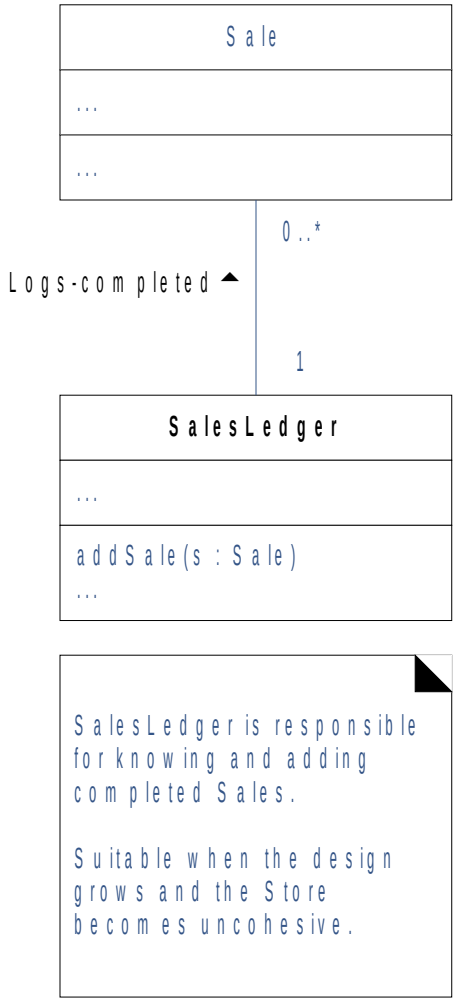
OBJECT DESIGN - makePayment



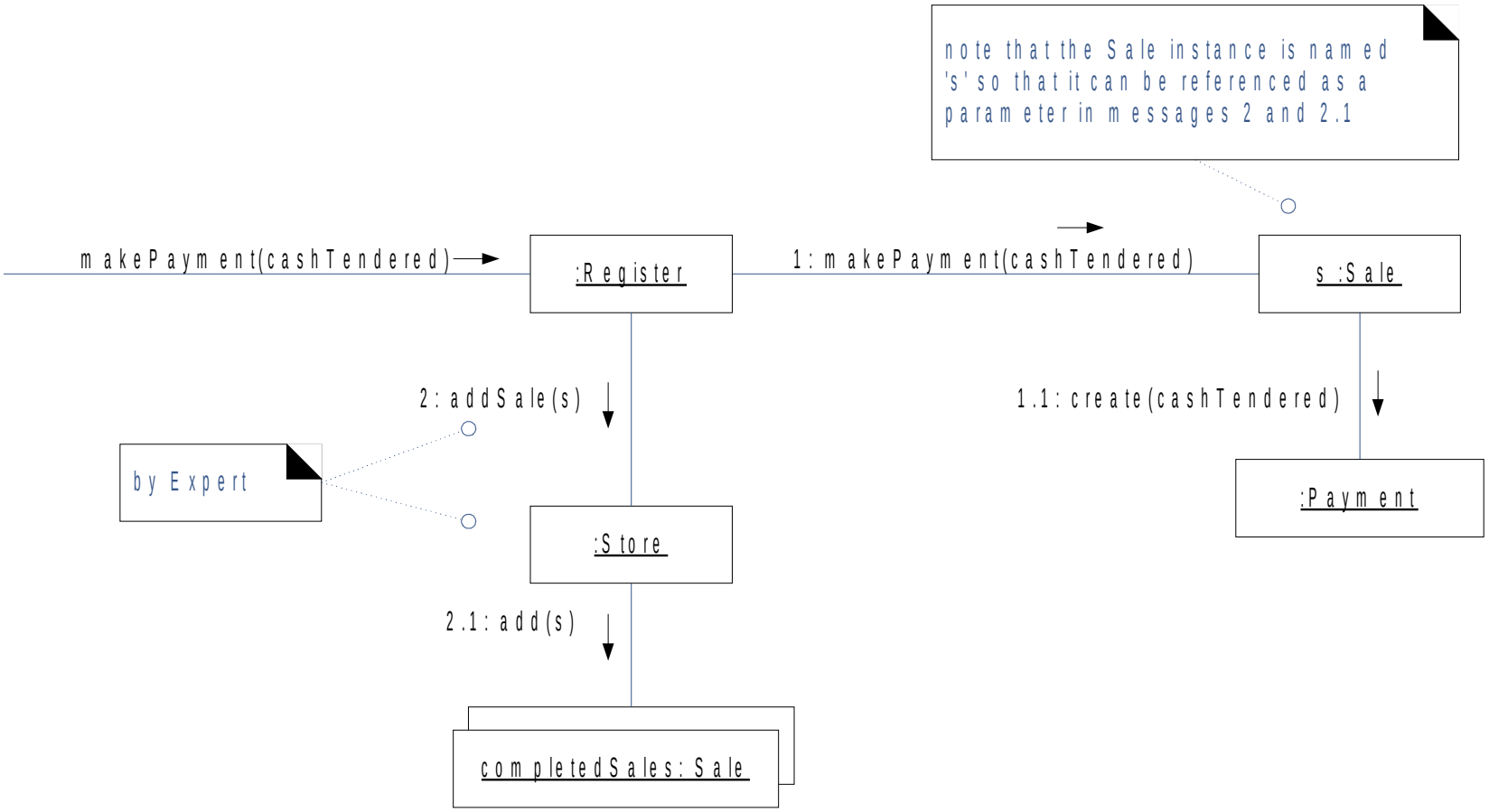
LOGGING THE SALE

- Who should be responsible for knowing the completed sales?





OBJECT DESIGN - LOGGING THE COMPLETED SALE

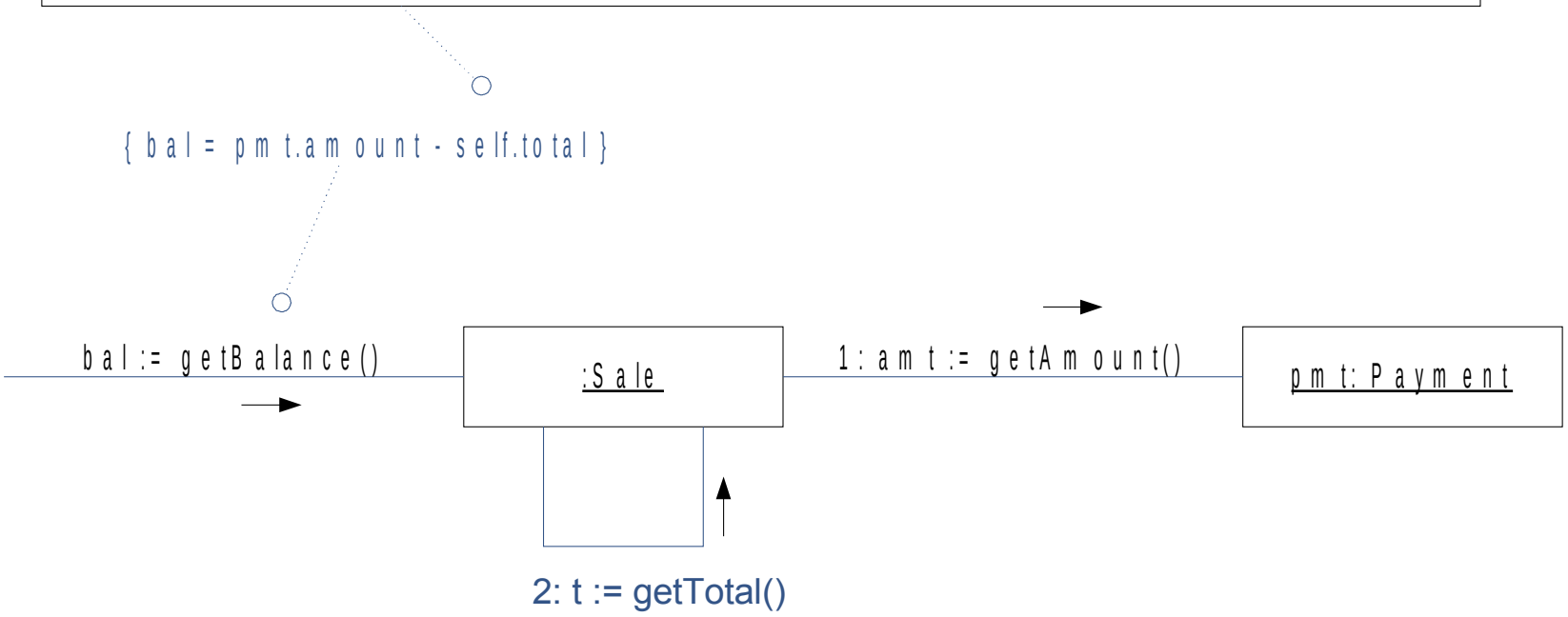


OBJECT DESIGN - CALCULATING THE BALANCE

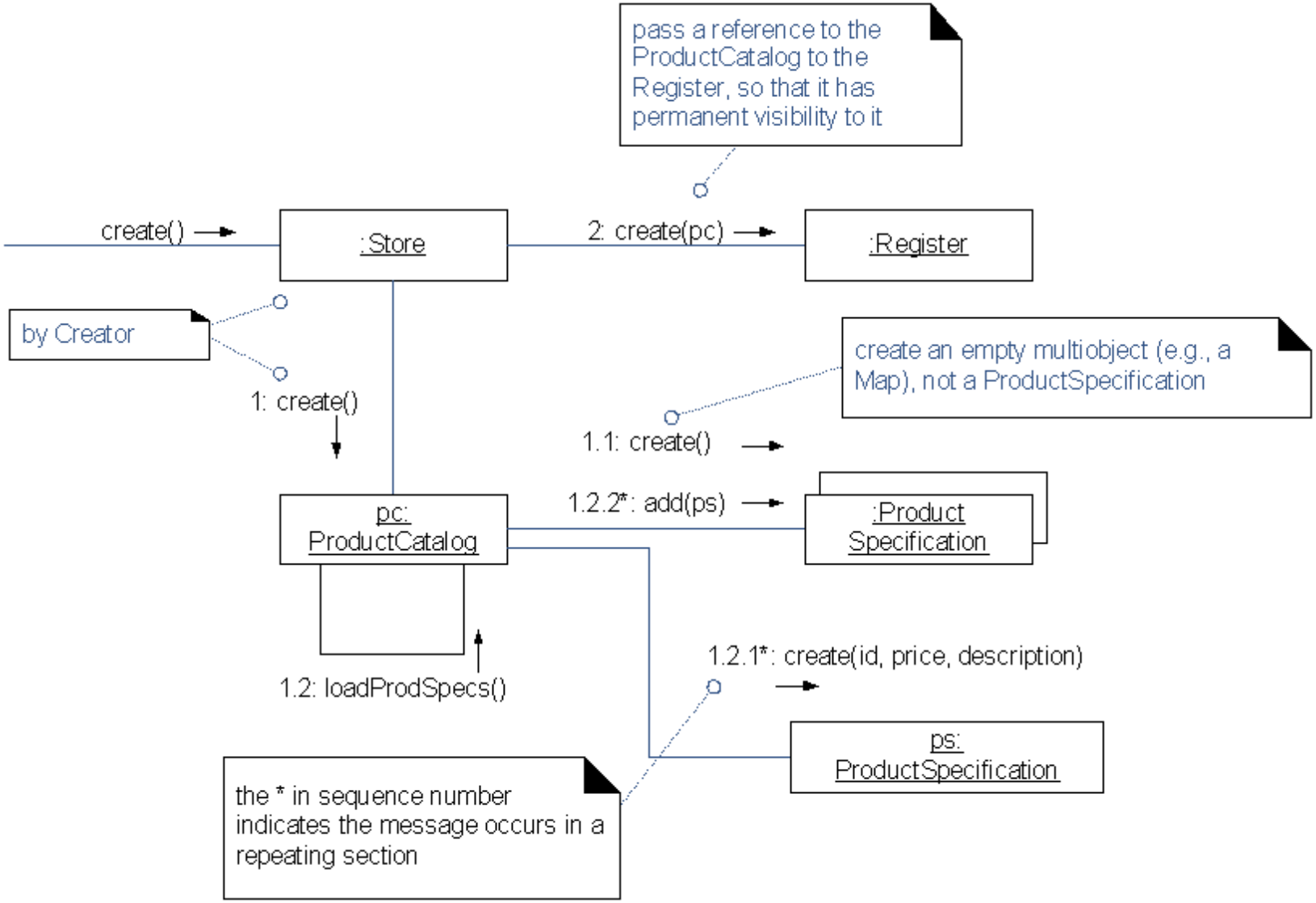
Note the use of "self" in the constraint. The formal OCL uses the special variable "self" for "this" (in Java and C++). "self" in this constraint implies the instance of the Sale.

Although official OCL is not being used, this style is borrowing from it.

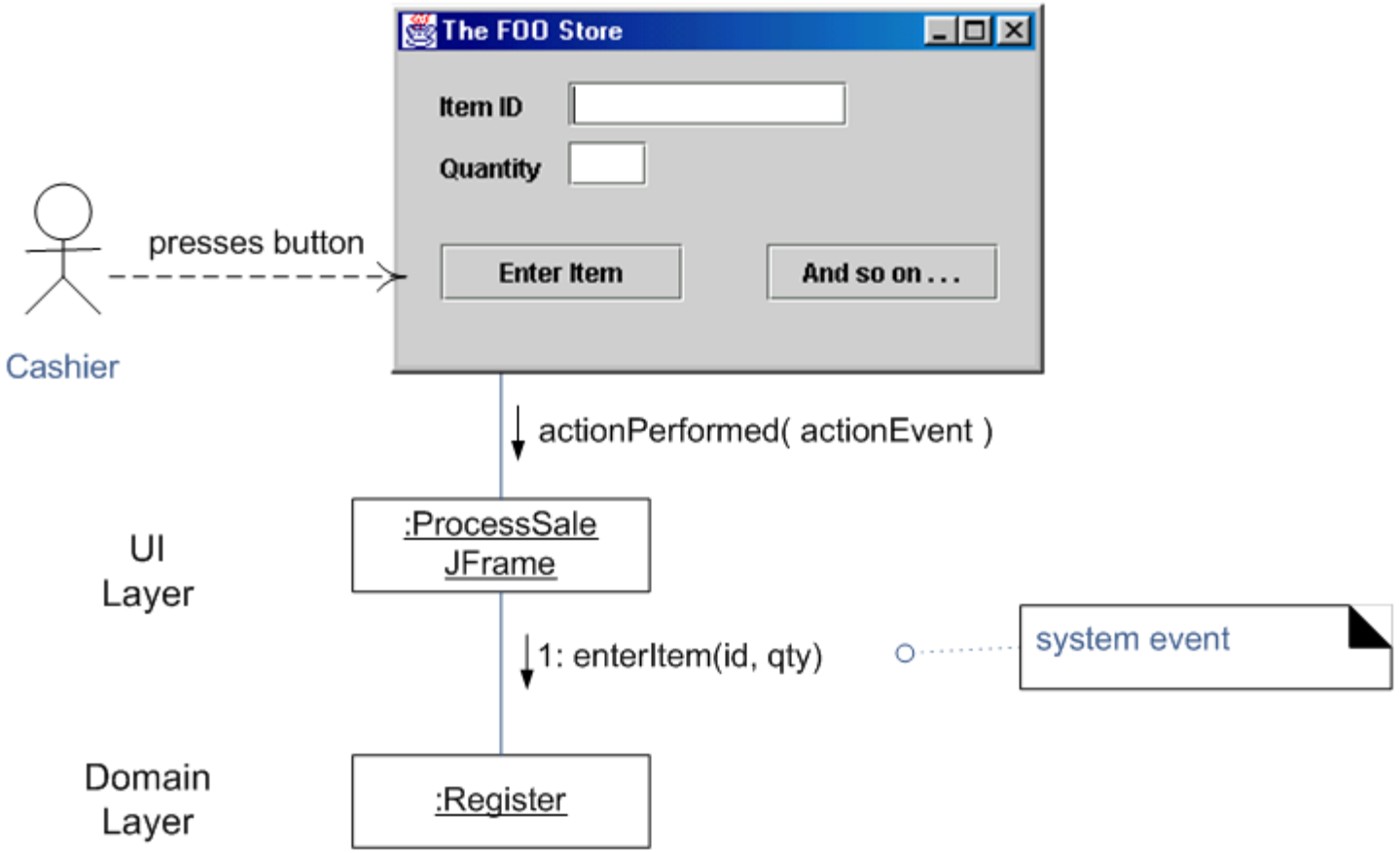
A constraint can be in any formal or informal language.



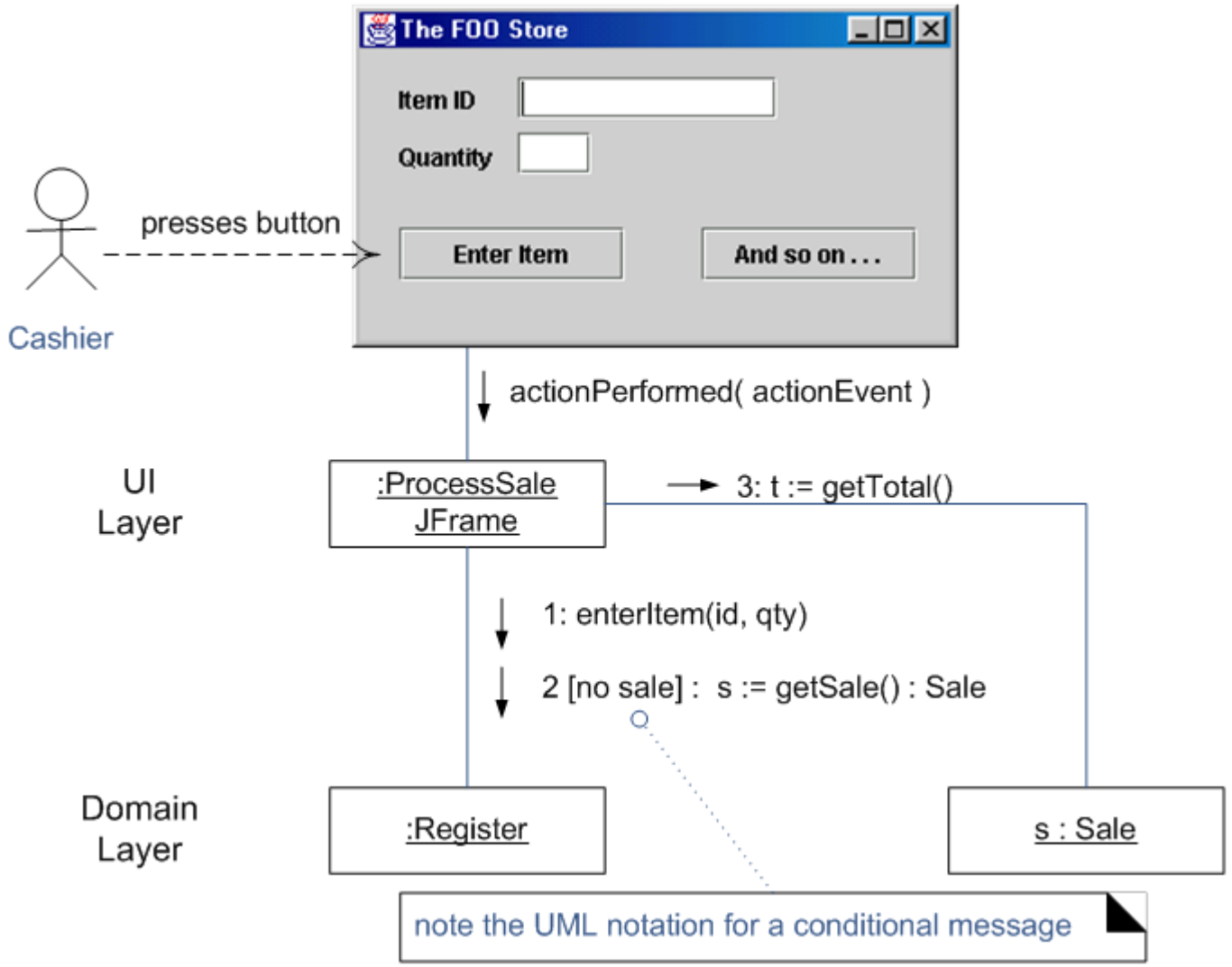
OBJECT DESIGN - startUp



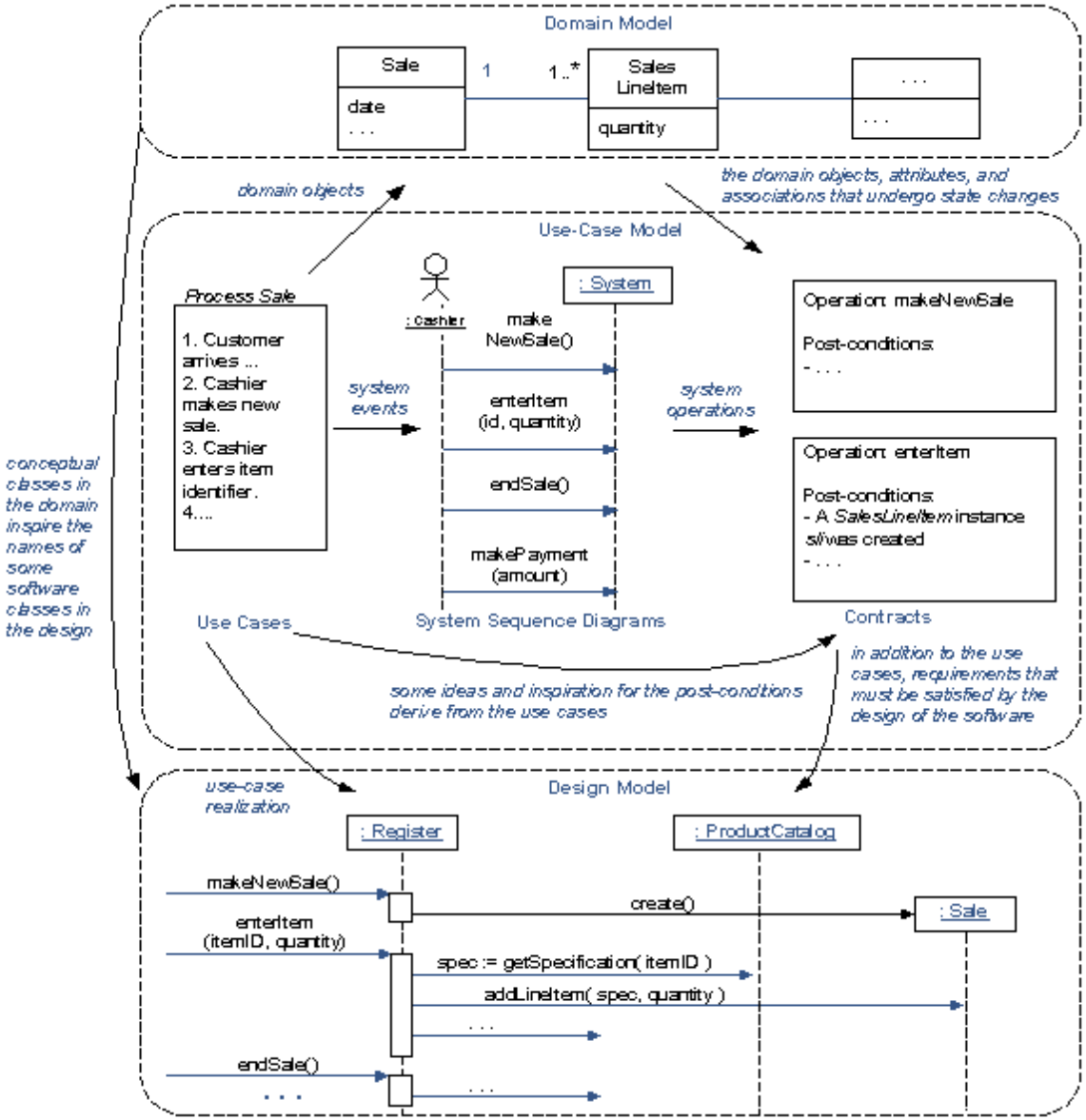
CONNECT UI LAYER AND DOMAIN LAYER



CONNECT UI LAYER AND DOMAIN LAYER

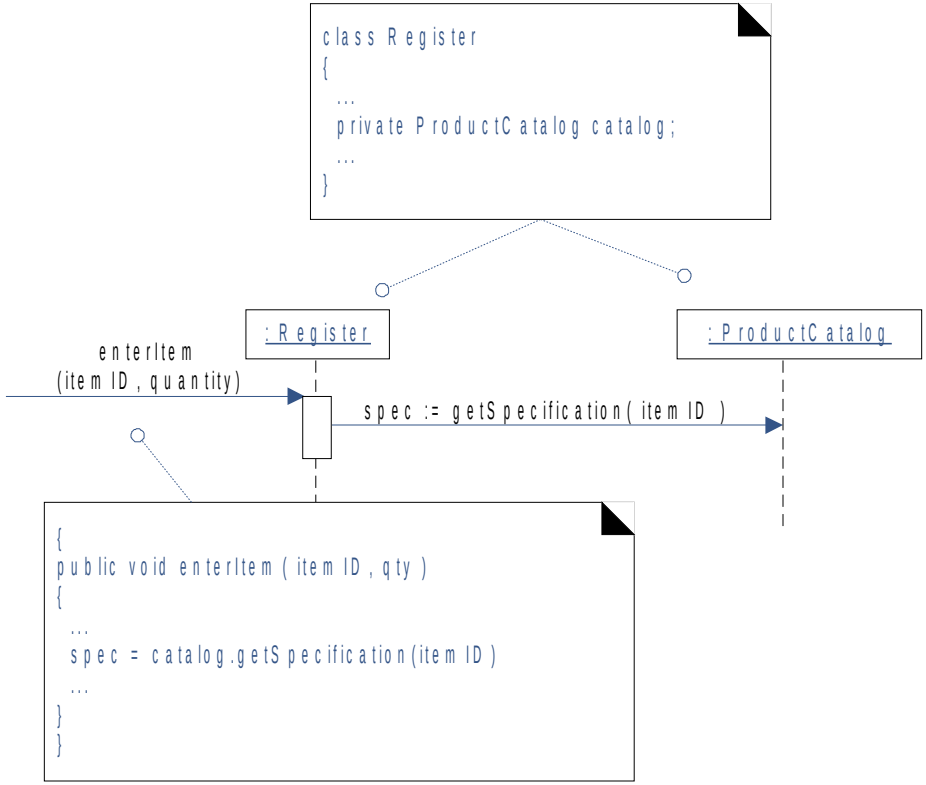


UP ARTIFACTS

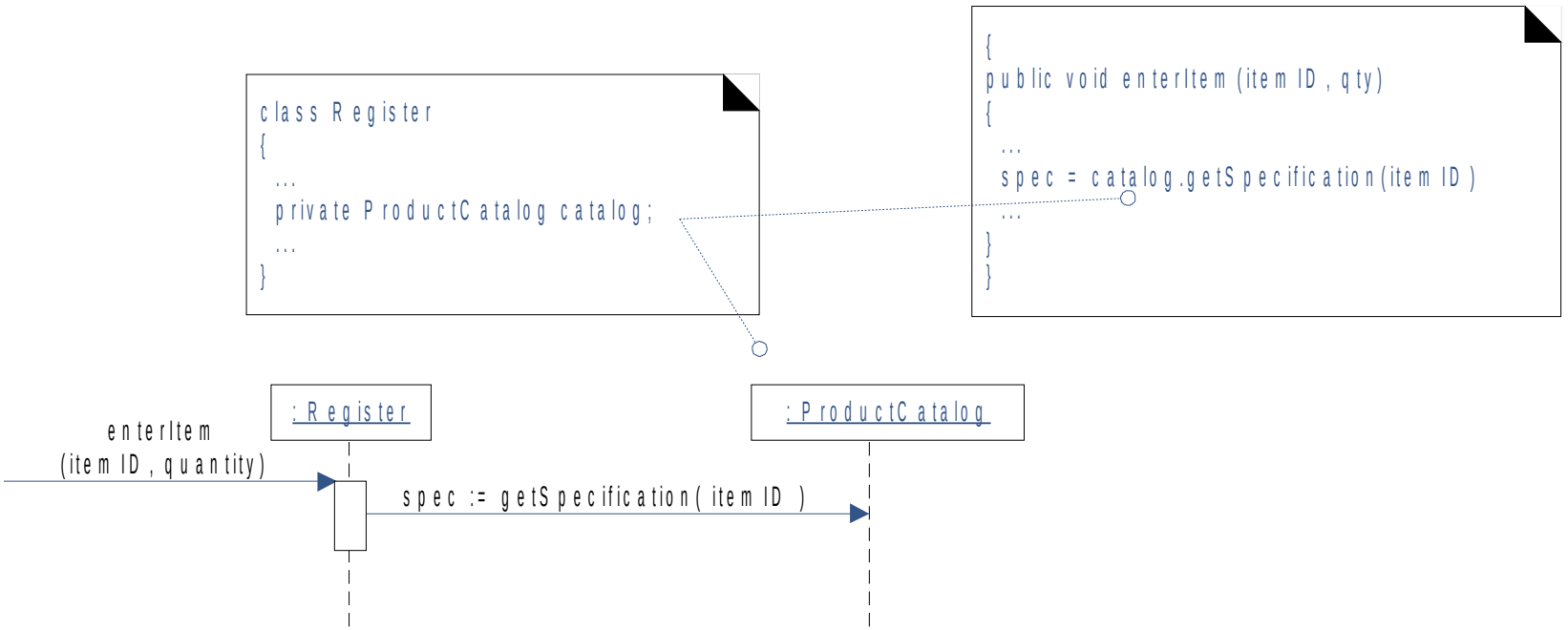


DETERMINING VISIBILITY

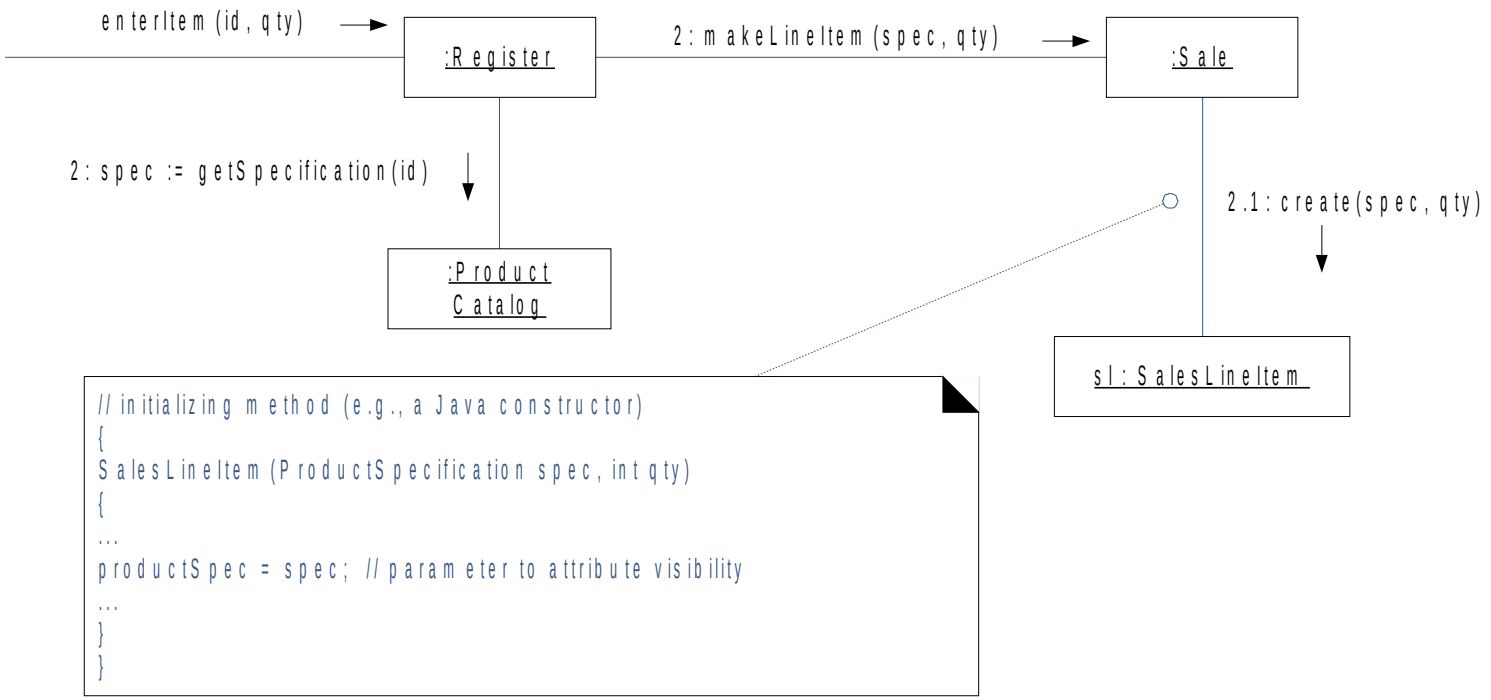
- Visibility - the ability of one object to see or have reference to another
- For a sender object to send a message a receiver object must be visible



ATTRIBUTE VISIBILITY

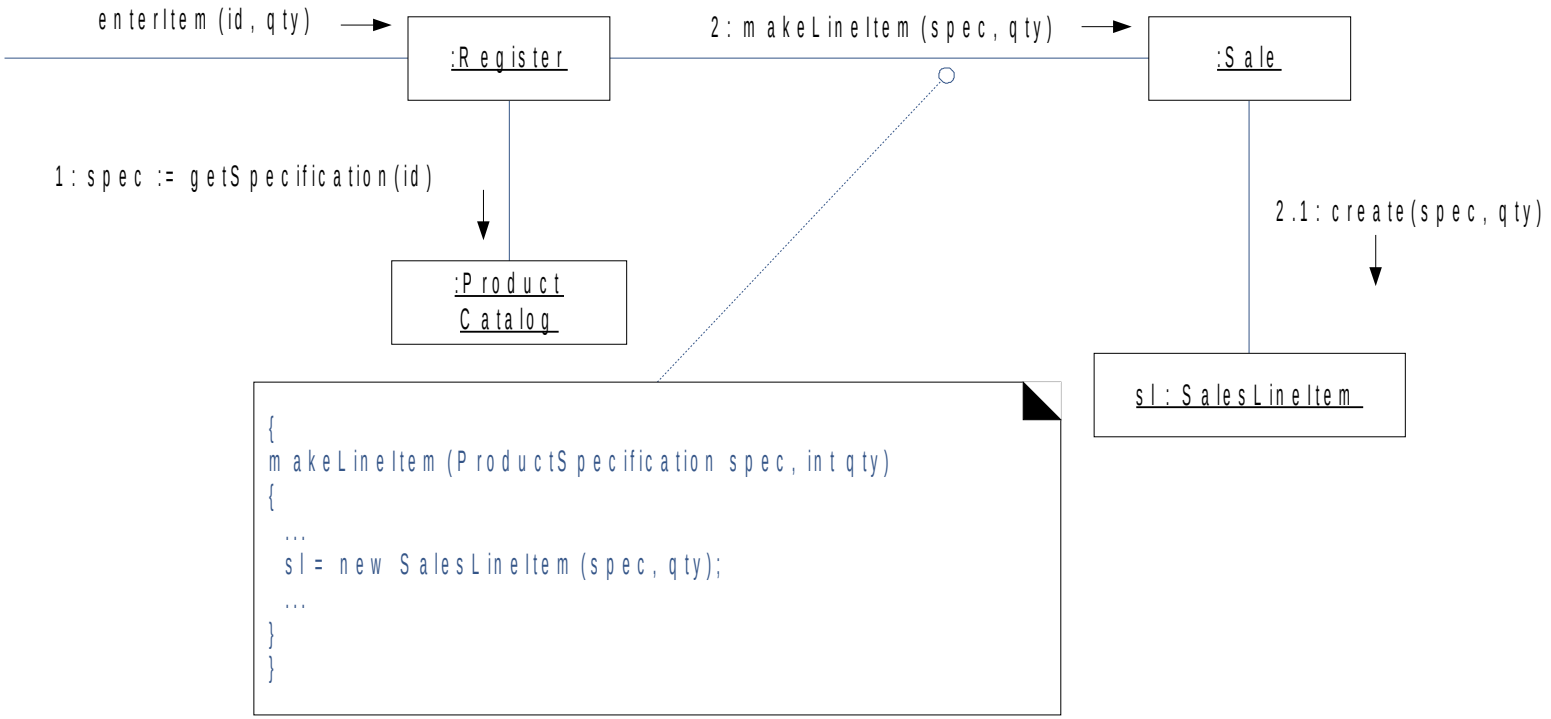


PARAMETER VISIBILITY



- Can be assigned to Attribute visibility

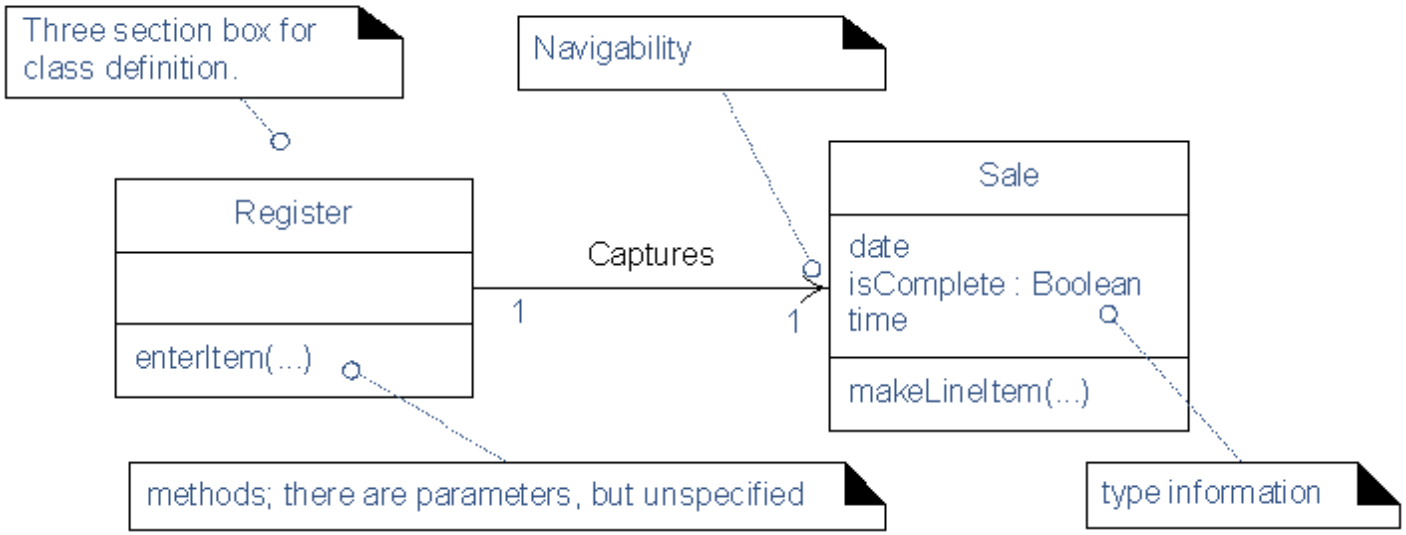
LOCAL VISIBILITY



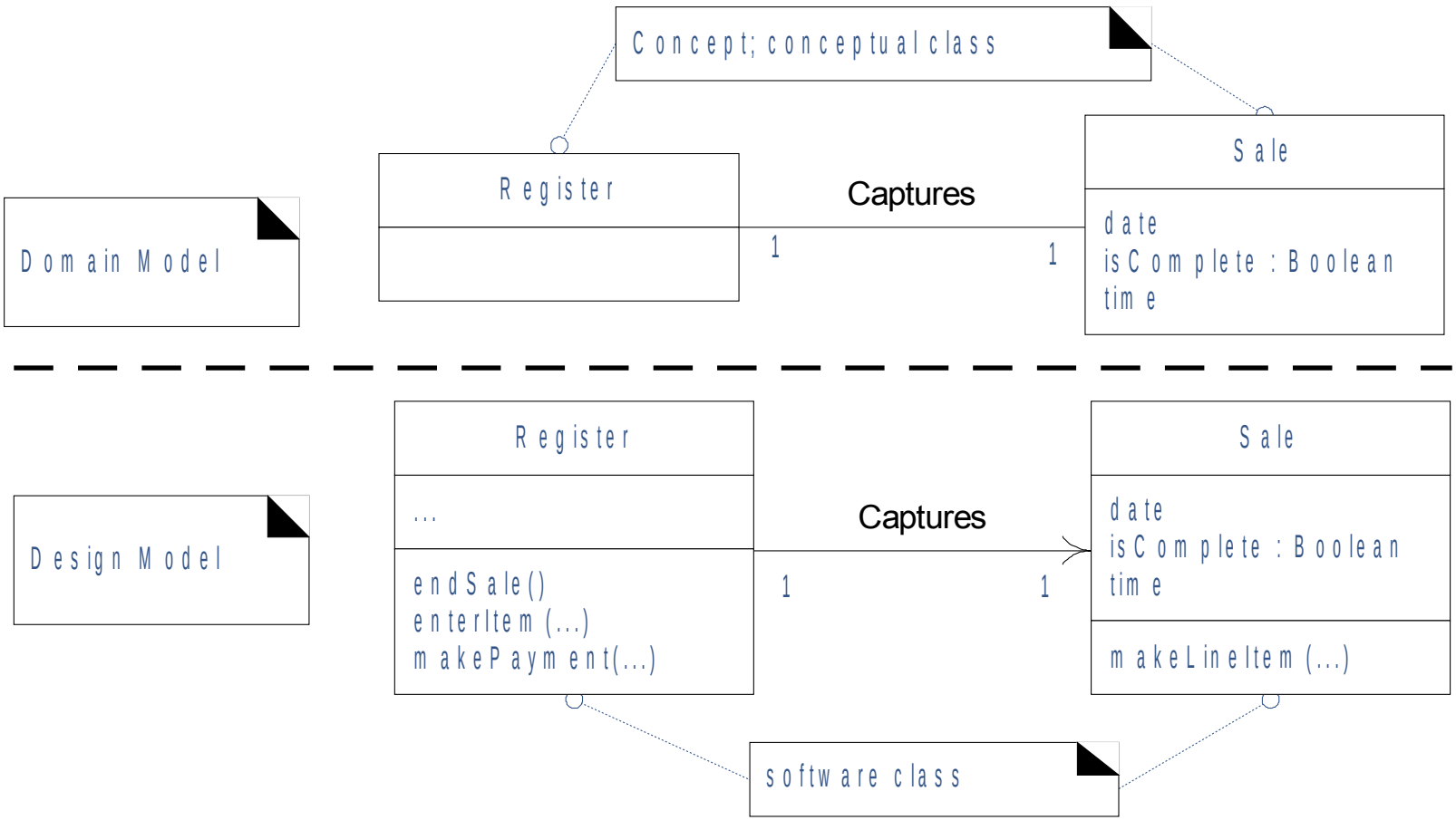
- Create a new local instance and assign it to a local variable
- Assign the returning object from a method invocation to a local variable
- Can be assigned to Attribute visibility

DESIGN CLASS DIAGRAMS

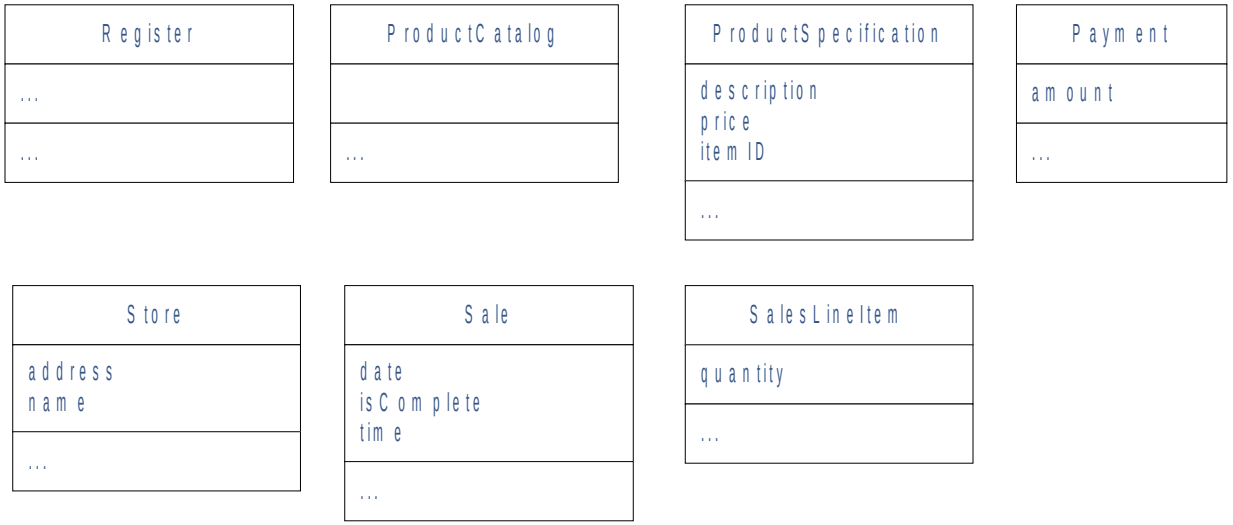
- Class diagram for design classes
  - Classes, associations, attributes
  - Interfaces, their operations and constants
  - Methods
  - Attribute type information
  - Navigability
  - Dependencies, generalization



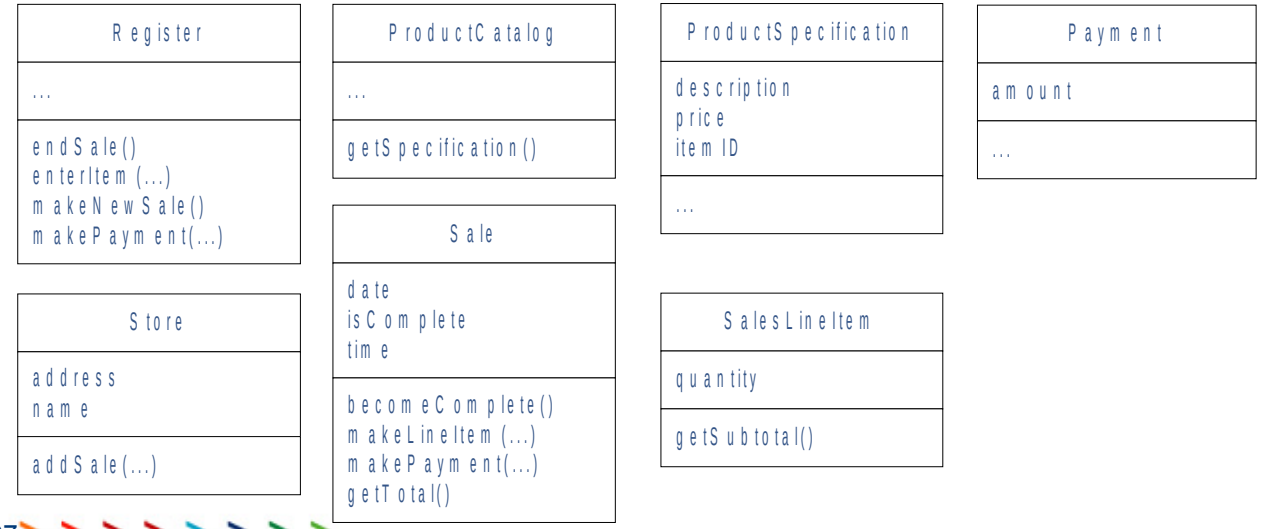
DOMAIN MODEL VS. DESIGN MODEL



POS DESIGN CLASSES

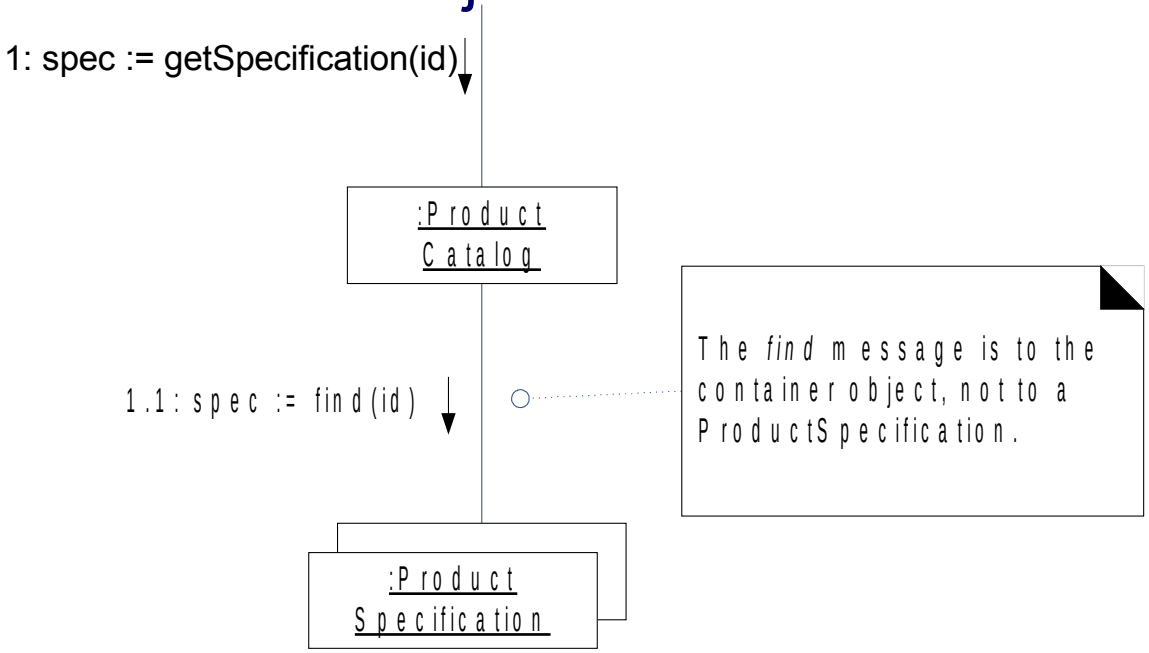


- Method names from interaction diagrams



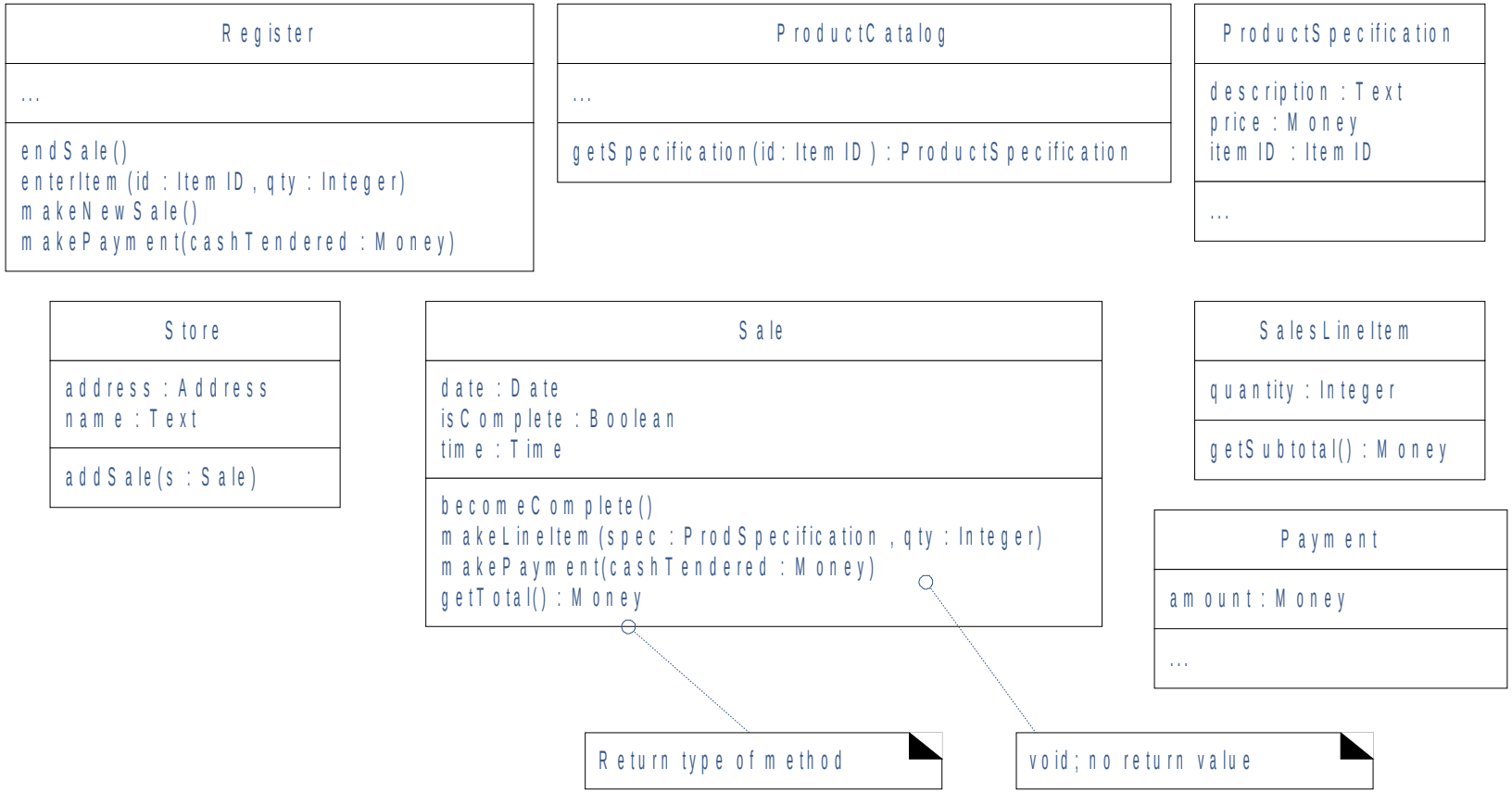
METHOD NAMES - MULTIOBJECTS

- A message to multiobject is a message to the container/collection object

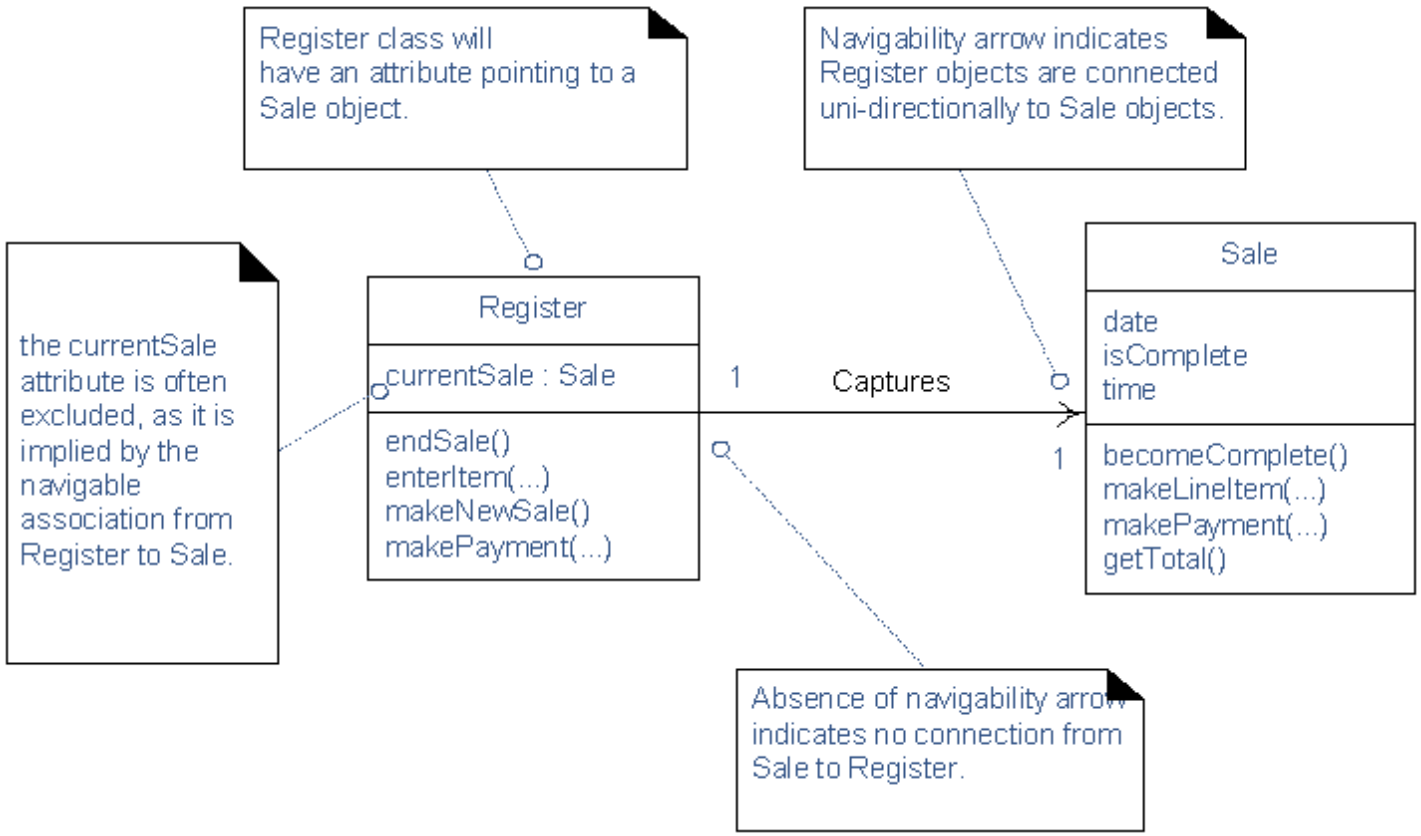


- The container/collection classes are often not shown explicitly in design class diagrams

ADDING TYPE INFORMATION



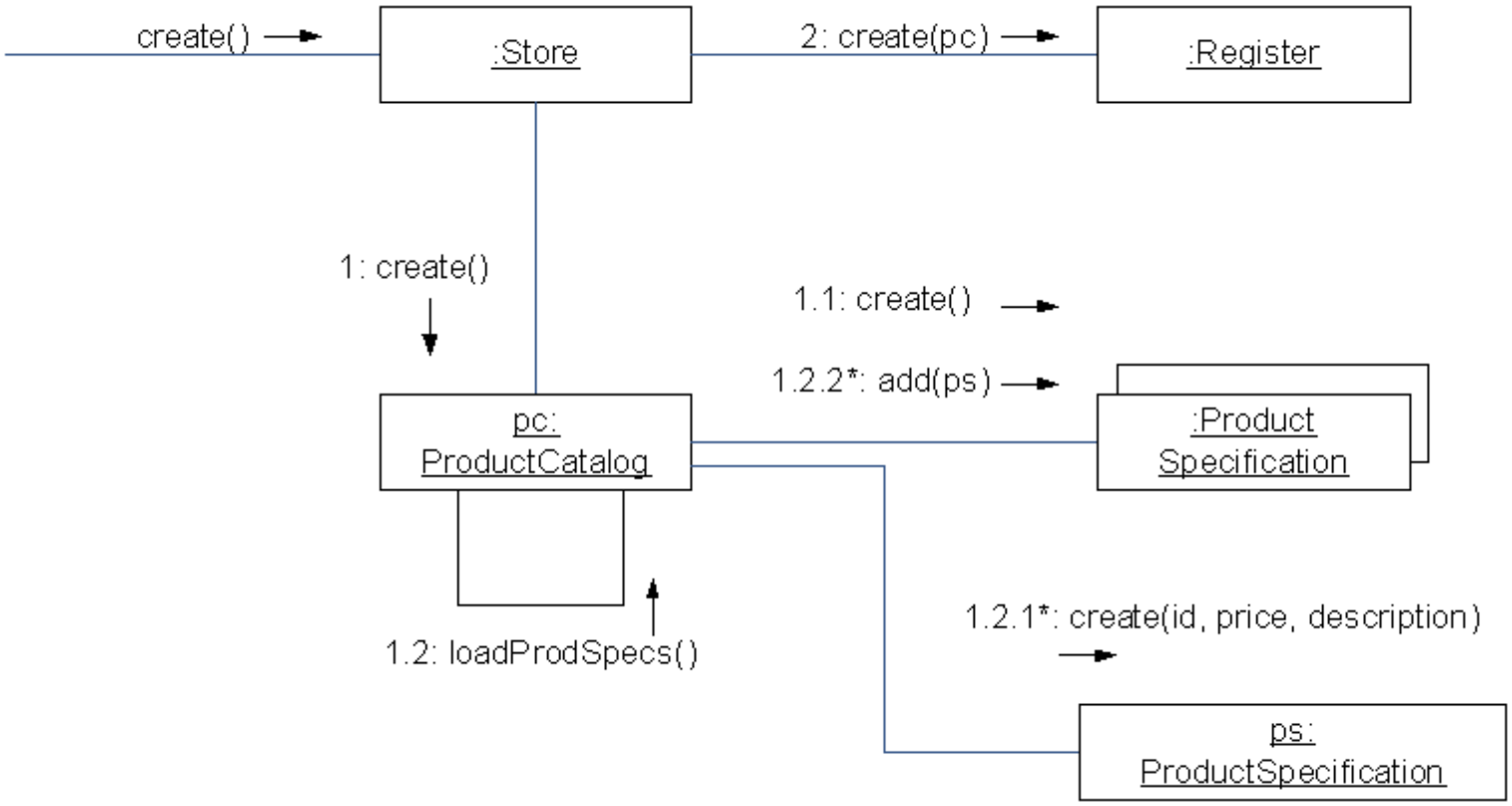
ADDING ASSOCIATIONS AND NAVIGABILITY



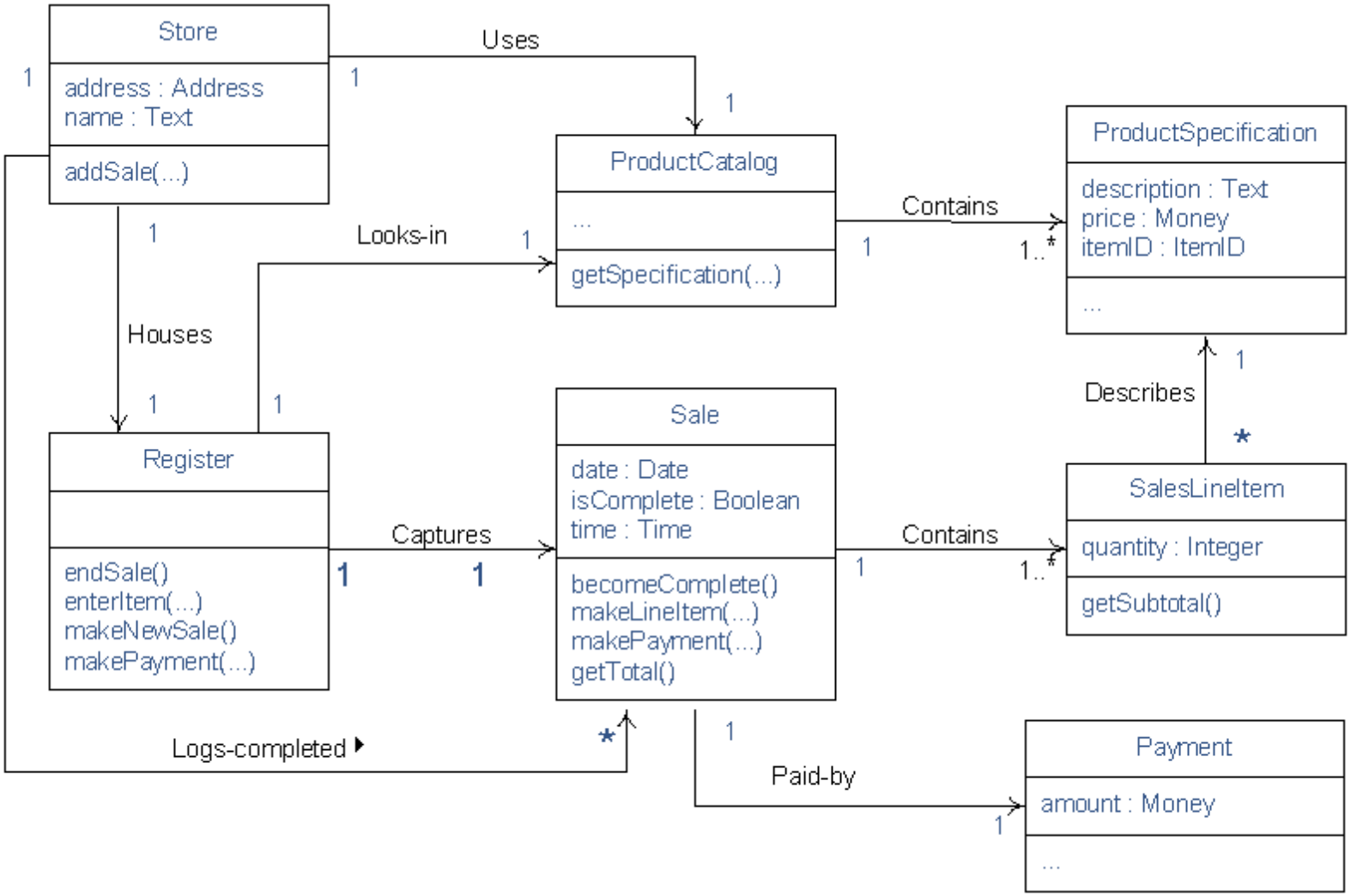
- Most associations in design class diagrams should be adorned with navigability arrows

WHERE TO FIND NAVIGABILITY?

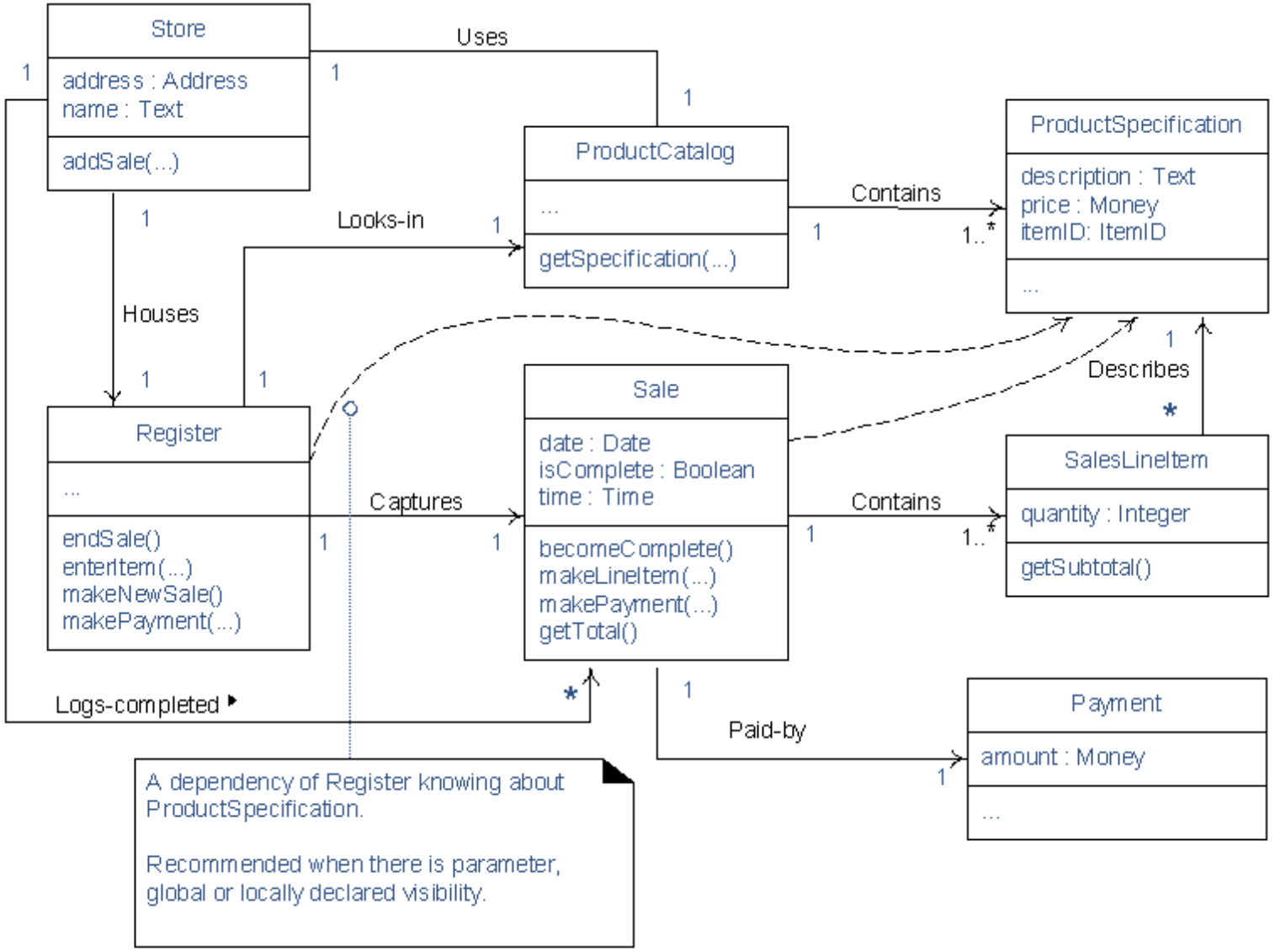
- Navigability is identified from the interaction diagrams



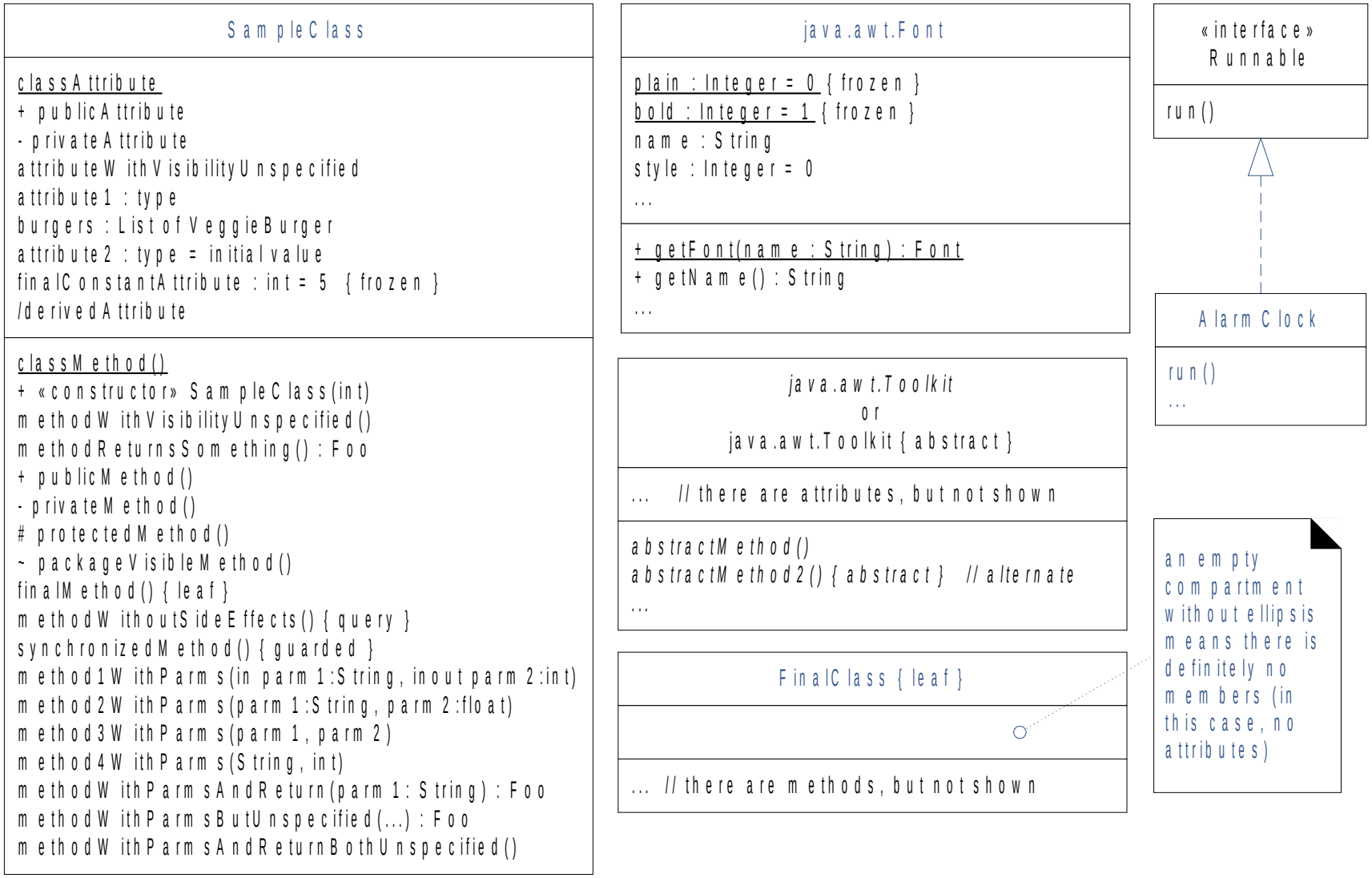
NAVIGABILITY



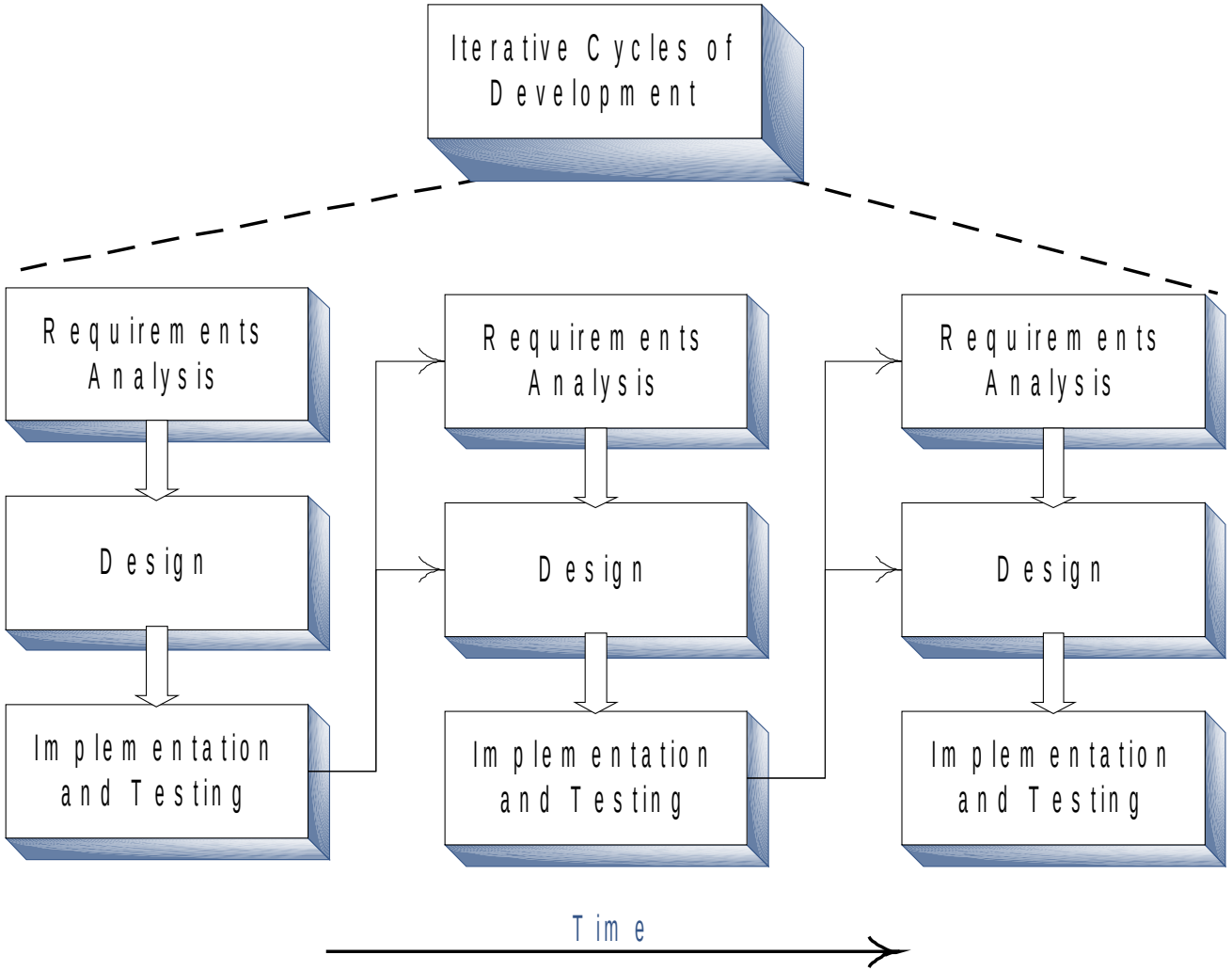
DEPENDENCY



NOTATION FOR MEMBER DETAILS



IMPLEMENTATION MODEL

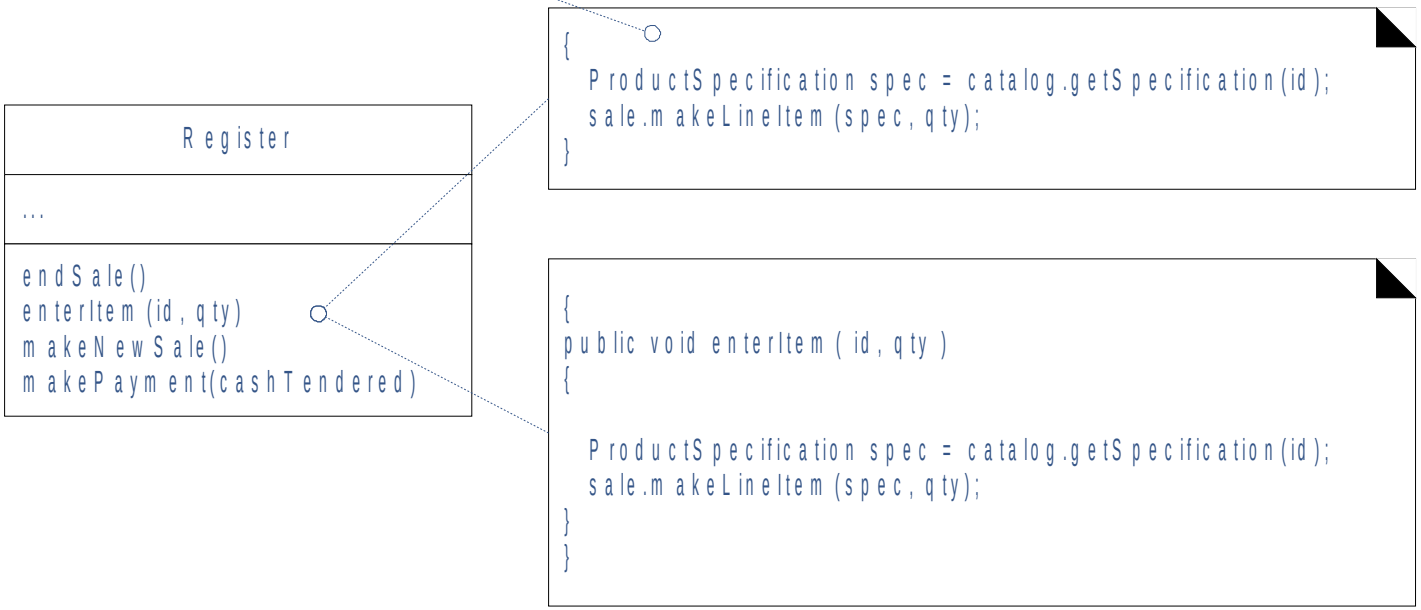


METHOD BODIES

UML notation:
A method body implementation may be shown in a UML note box. It should be placed within braces, which signifies it is semantic influence (it is more than just a comment).

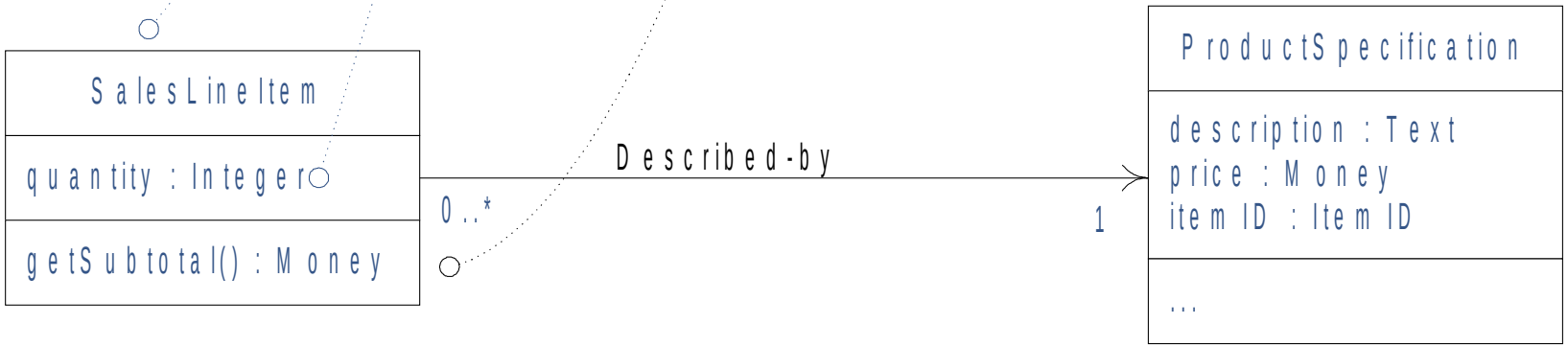
The synax may be pseudo-code, or any language.

It is common to exclude the method signature (public void ...), but it is legal to include it.

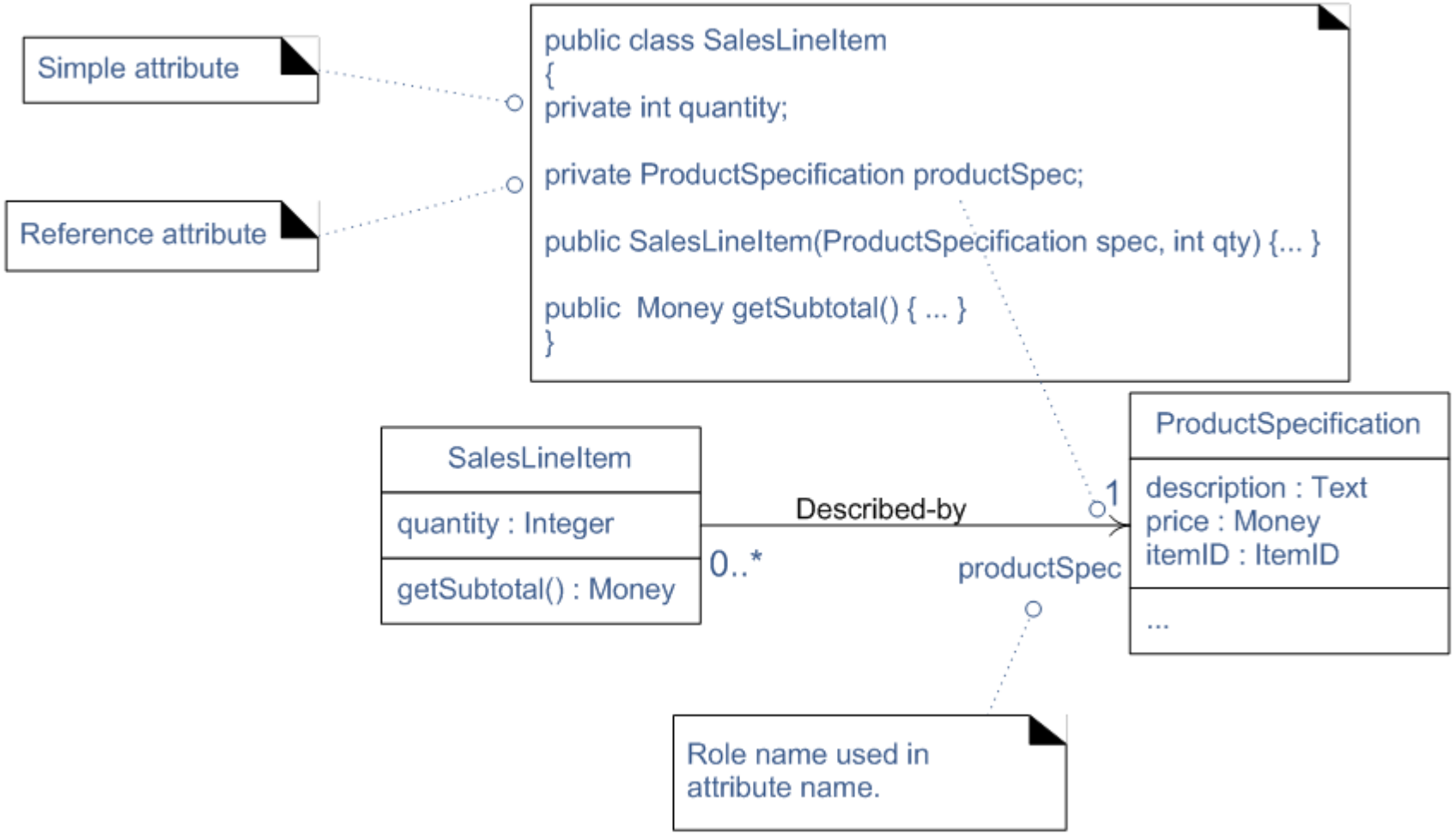


CLASS DEFINITIONS

```
public class SalesLineItem
{
private int quantity;

public SalesLineItem (ProductSpecification spec, int qty) { ... }

public Money getSubtotal() { ... }

}
```



CLASS DEFINITIONS II



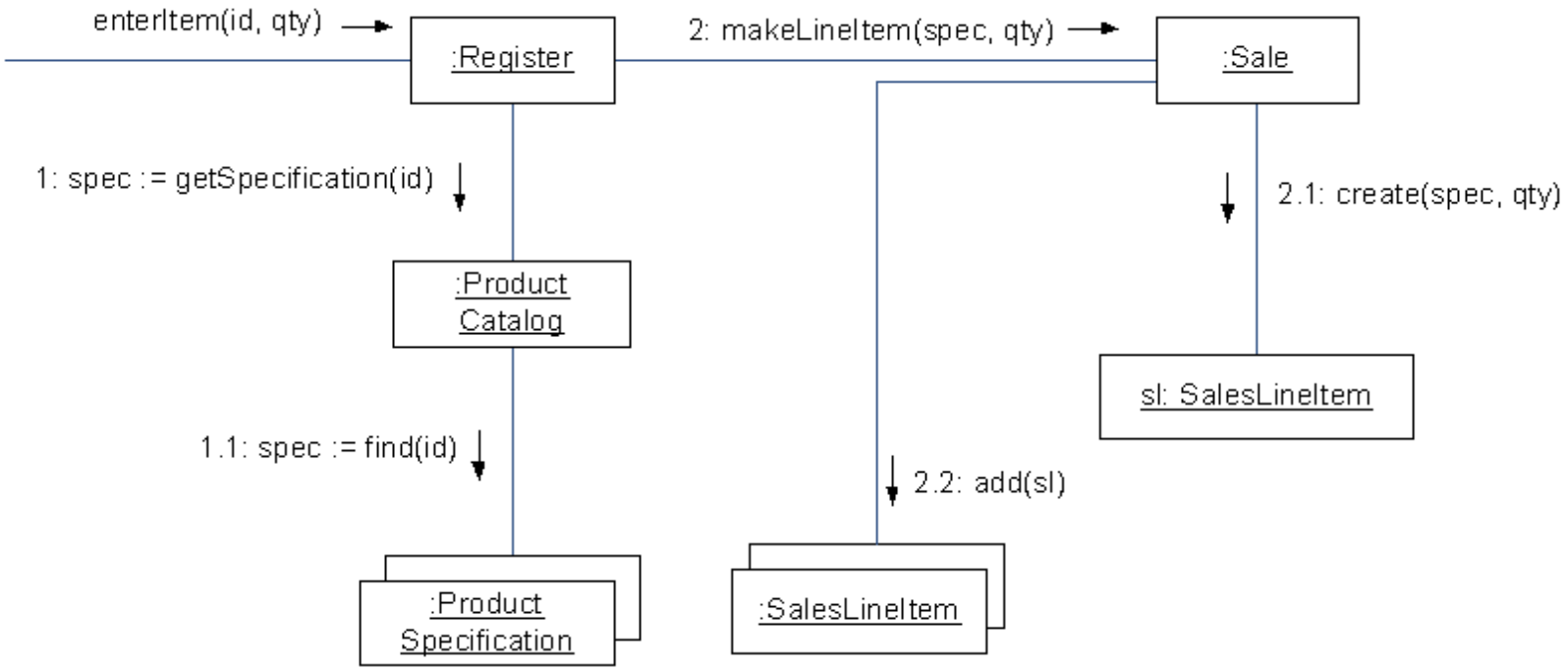
MAPPING ATTRIBUTES

```
public class Sale
{
private Date dateTime = new Date();
...
}
```

Sale
date : Date isComplete : Boolean time : Time
becomeComplete() makeLineItem() makePayment() getTotal()

In Java, the java.util.Date class combines both date and timestamp information. Therefore, the separate attributes in the design can be collapsed when mapping to Java.

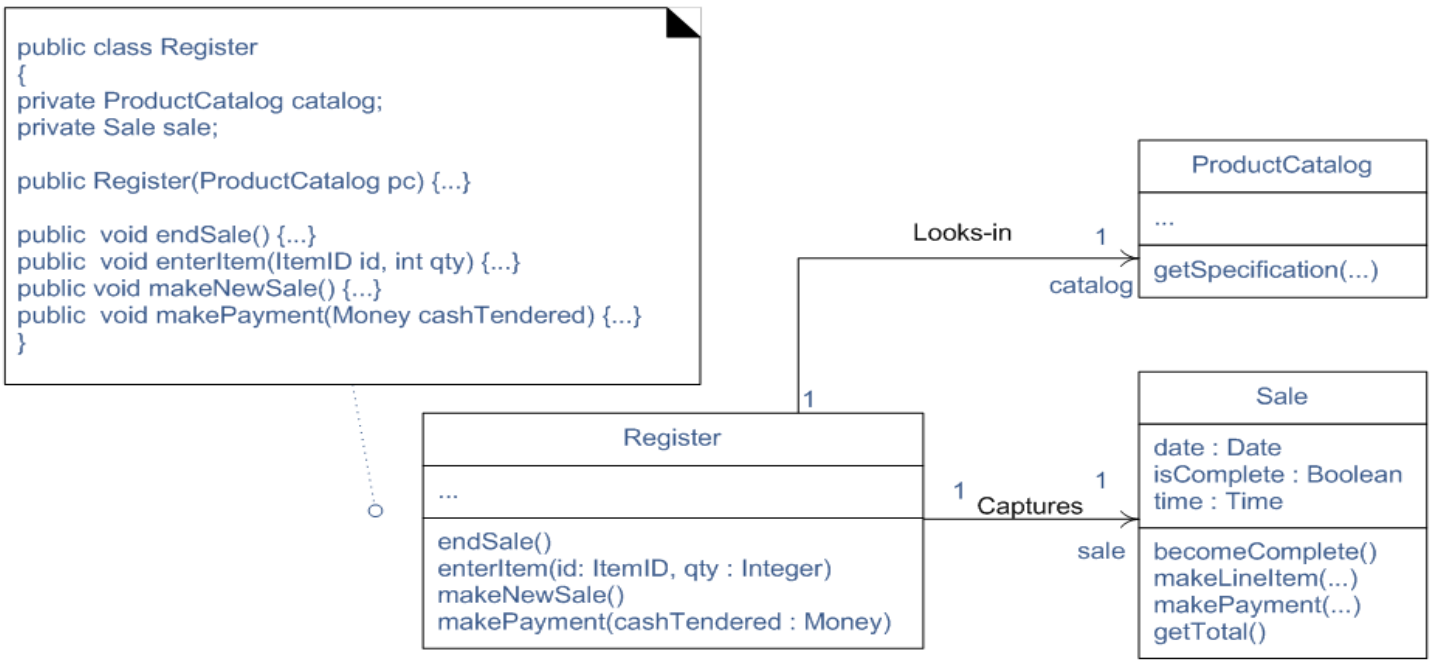
CREATING METHODS



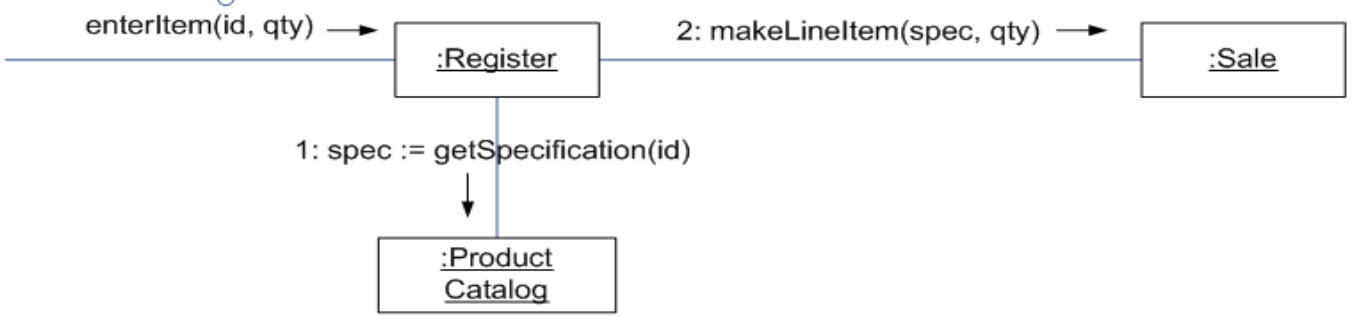
- The events from collaboration / sequence diagrams shall become methods on class diagrams

CREATING METHODS II

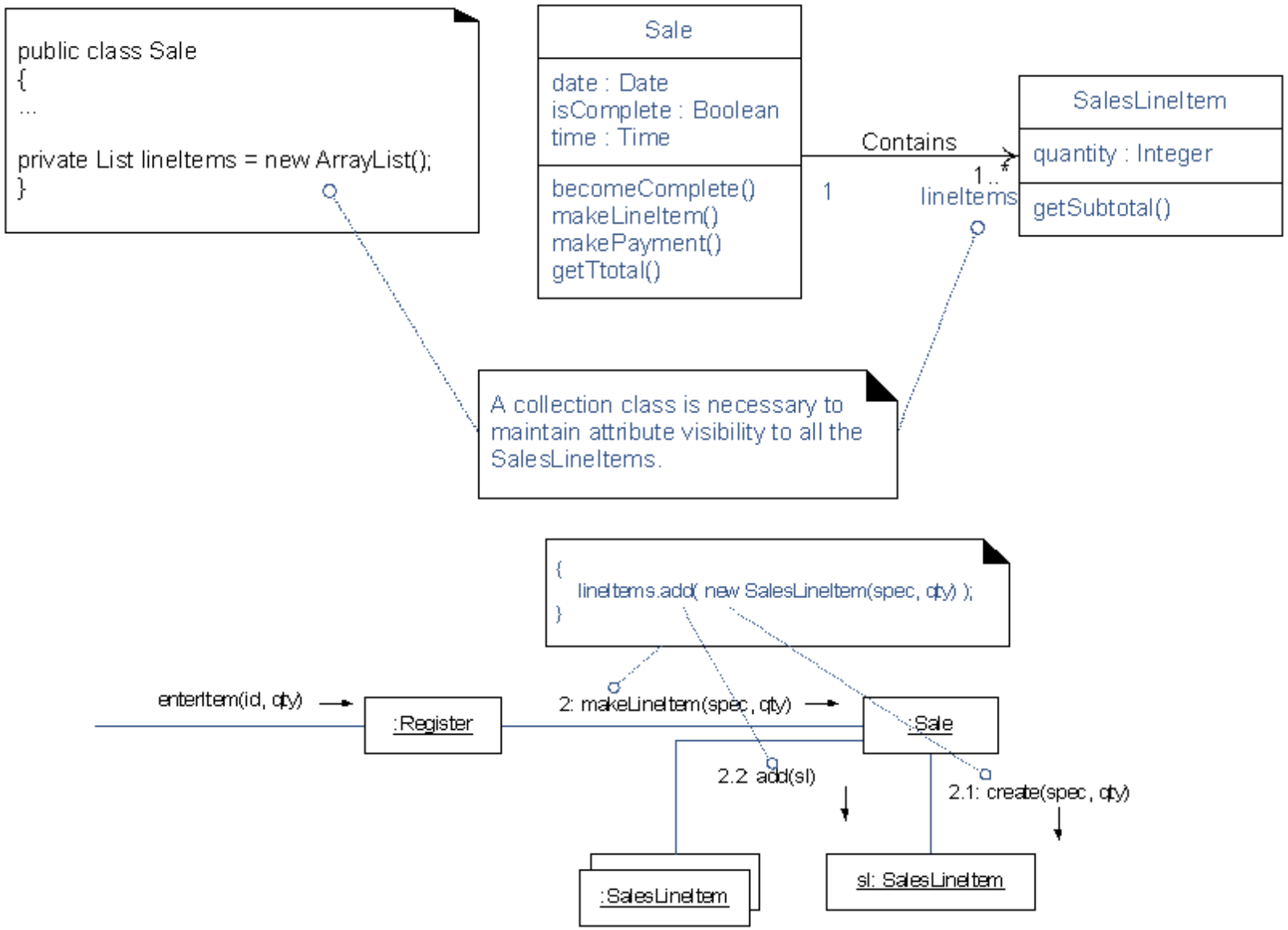
```
public class Register
{
private ProductCatalog catalog;
private Sale sale;

public Register(ProductCatalog pc) {...}

public  void endSale() {...}
public  void enterItem(ItemID id, int qty) {...}
public void makeNewSale() {...}
public  void makePayment(Money cashTendered) {...}
}
```



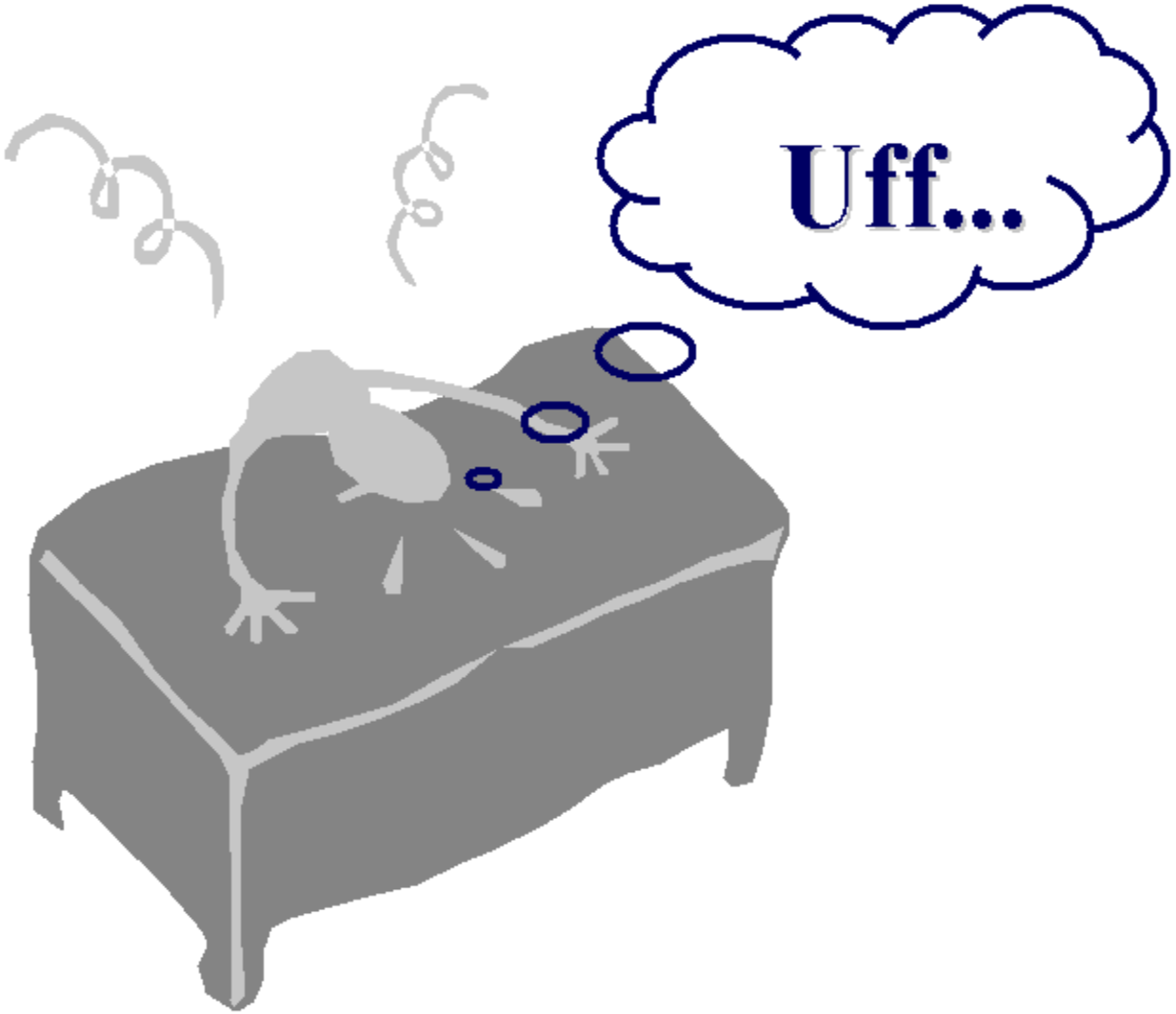

```
{
   ProductSpecification spec = catalog.getSpecification(id);
   sale.makeLineItem(spec, qty);
}
```



ADDING COLLECTIONS



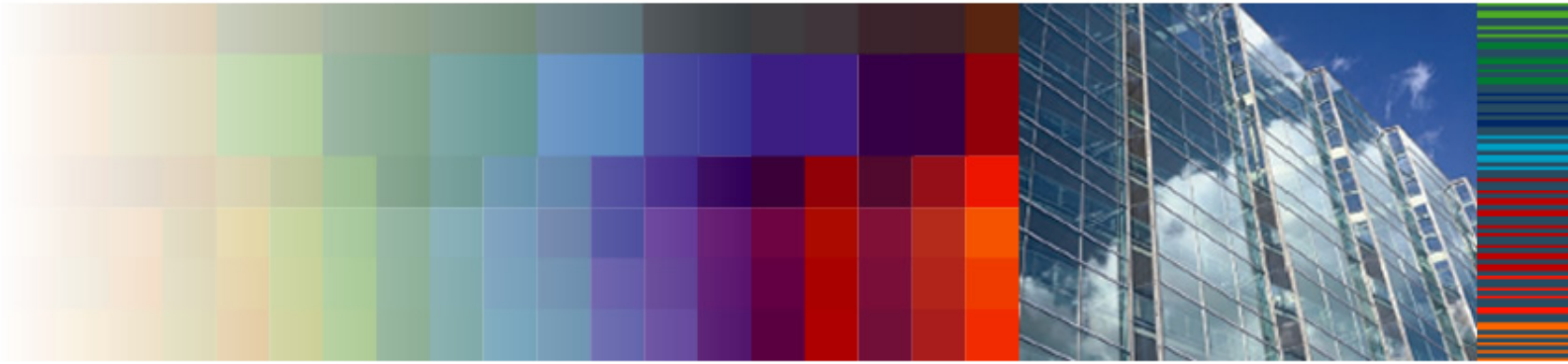
Q&A



RESOURCES

- Applying UML and Patterns, Craig Larman Prentice Hall 2002, ISBN 0130925691

- Rational Unified Process, IBM 2003



UNICORN

TAK SE DĚLÁ SOFTWARE