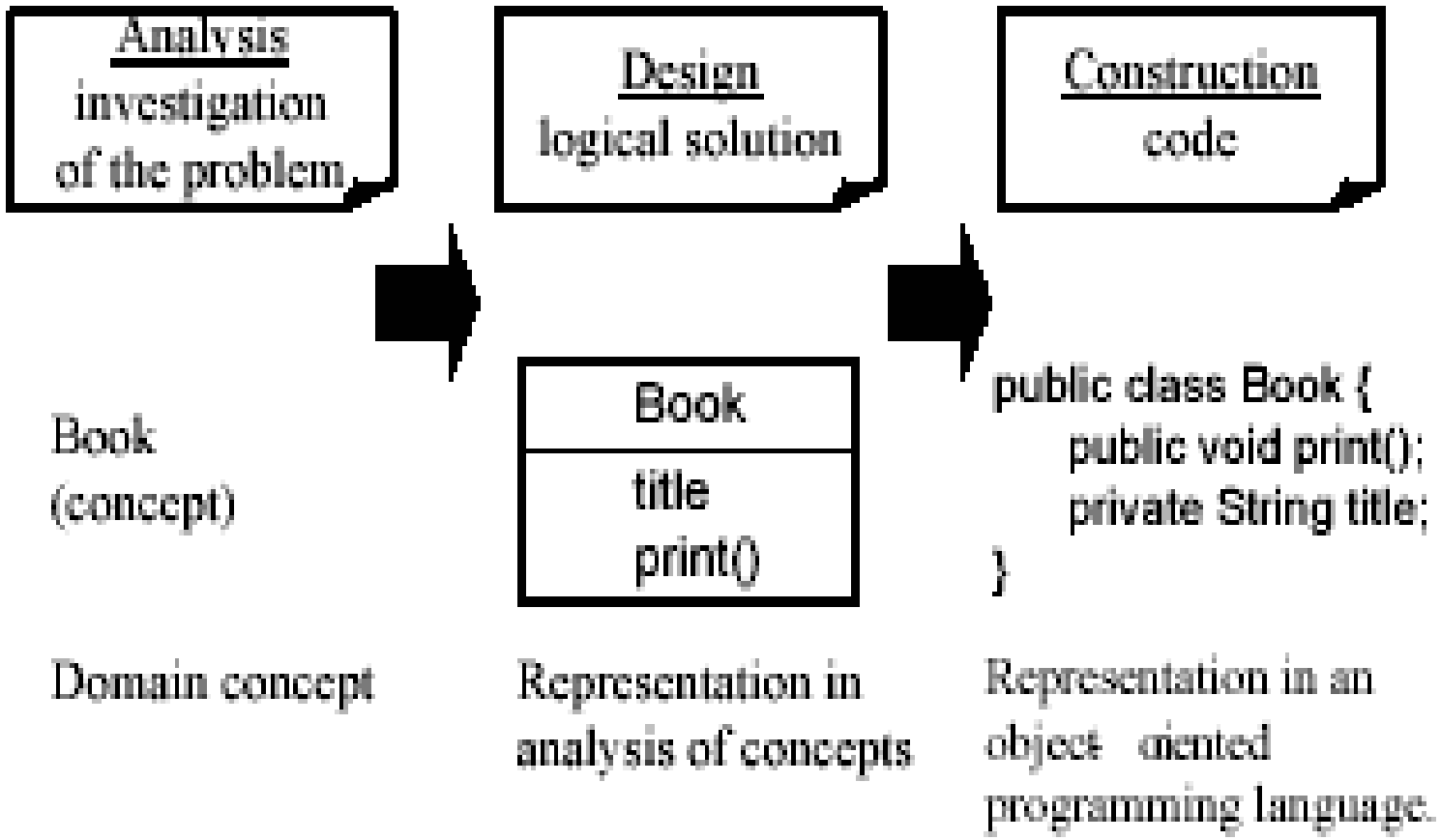
Chapter 1

Applying UML and Patterns

The Need for Software Blueprints

- Knowing an object-oriented language and having access to a library is necessary but not sufficient in order to create object software.
- In between a nice idea and a working software, there is much more than programming.
- Analysis and design provide software “blueprints”, illustrated by a modeling language, like the Unified Modeling Language (UML).
- Blueprints serve as a tool for thought and as a form of communication with others.

From Design to Implementation



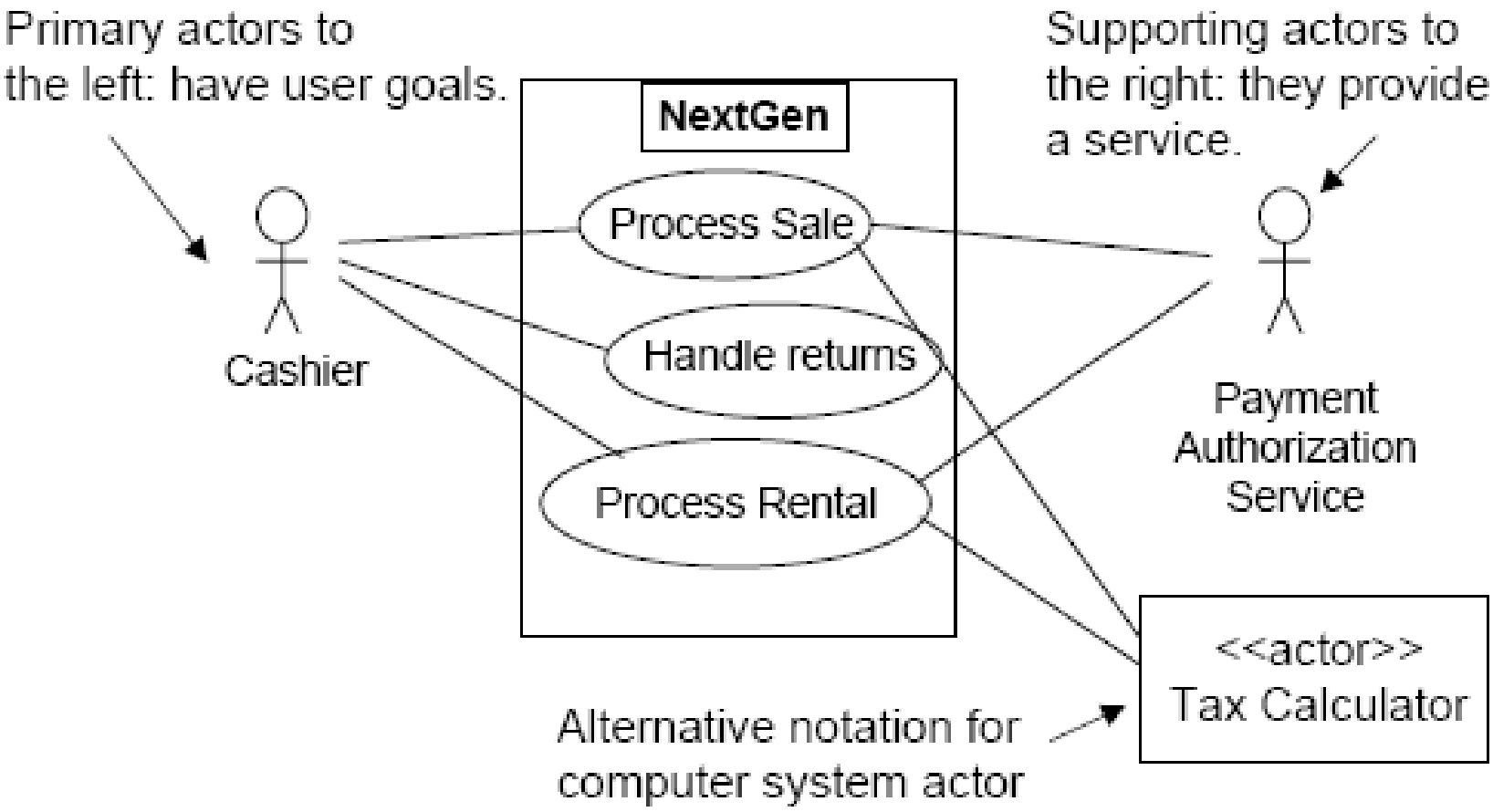
Introduction to Requirements

- Requirements are system capabilities and conditions to which the system must conform.
- Functional requirements
  - Features and capabilities.
  - Recorded in the Use Case model (see next), and in the systems features list of the Vision artifact.
- Non-functional (or quality requirements)
  - Usability (Help, documentation, …), Reliability (Frequency of failure, recoverability, …), Performance (Response times, availability, …), Supportability (Adaptability, maintainability, …)
  - Recorded in the Use Case model or in the Supplementary Specifications artifact.

Use-Case Model: Writing Requirements in Context
Use cases and adding value

- Actor: something with behavior, such as a person, computer system, or organization, e.g. a cashier.
- Scenario: specific sequence of actions and interactions between actors and the system under discussion, e.g. the scenario of successfully purchasing items with cash.
- Use case: a collection of related success and failure scenarios that describe actors using a system to support a goal.

Use Case Diagrams



Use-Case Model: Drawing System Sequence Diagrams

System Behavior and UML Sequence Diagrams

- It is useful to investigate and define the behavior of the software as a “black box”.
- System behavior is a description of *what the system does* (without an explanation of how it does it).
- Use cases describe how external actors interact with the software system. During this interaction, an actor generates events.
- A request event initiates an operation upon the system.

System Behavior and System Sequence Diagrams (SSDs)

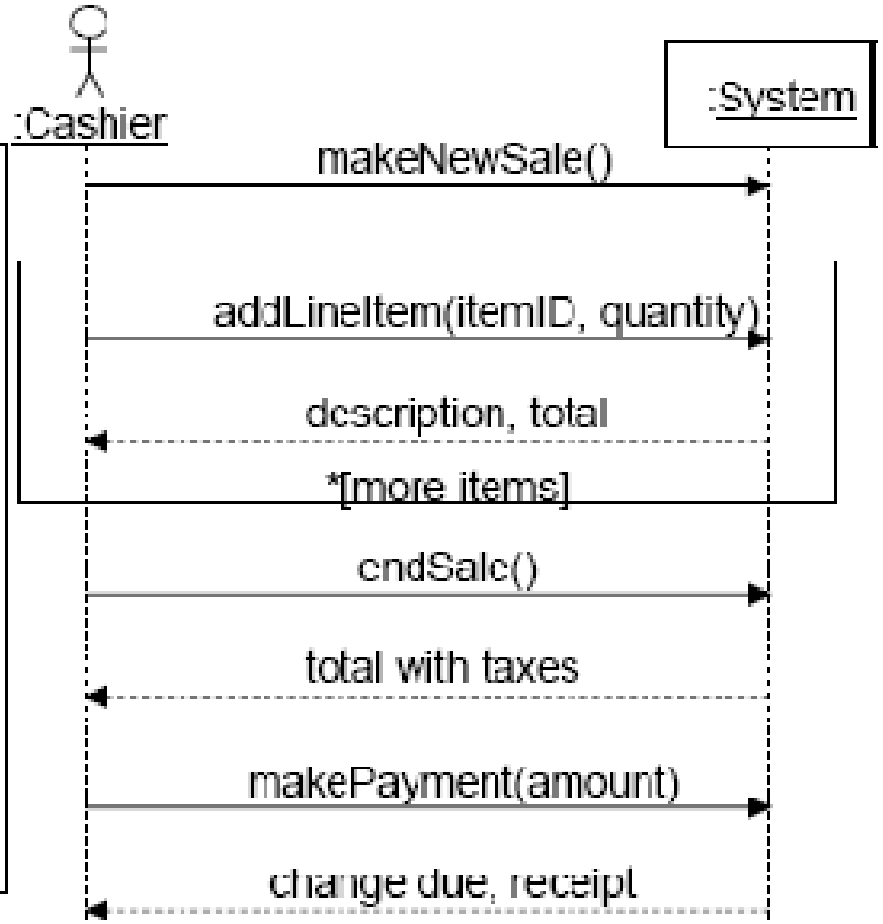
- A sequence diagram is a picture that shows, for a particular scenario of a use case, the events that external actors generate, their order, and possible inter-system events.
- All systems are treated as a black box; the diagram places emphasis on events that cross the system boundary from actors to systems.

SSD and Use Cases

Simple cash-only Process Sale Scenario

1. Customer arrives at a POS checkout with goods 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.

...



Naming System Events and Operations

The set of all required system operations is determined by identifying the system events.

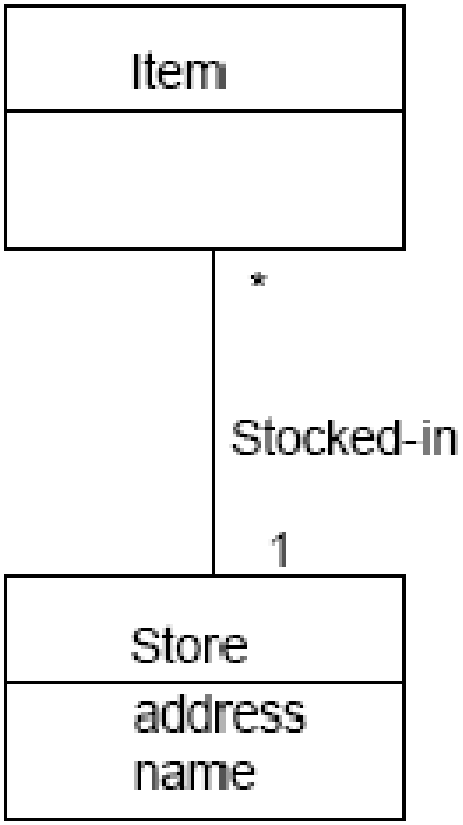
- makeNewSale()
- addLineItem(itemID, quantity)
- endSale()
- makePayment(amount)

Domain Model: Visualizing Concepts

Domain Models

- A Domain Model illustrates meaningful concepts in a problem domain.
- It is a representation of real-world things, not software components.
- It is a set of static structure diagrams; no operations are defined.
- It may show:
  - concepts
  - associations between concepts
  - attributes of concepts

Domain Models



- A Domain Model is a description of things in the real world.
- A Domain Model is not a description of the software design.
- A concept is an idea, thing, or object.

Strategies to Identify Conceptual Classes

- Use noun phrase identification.
  - Identify noun (and noun phrases) in textual descriptions of the problem domain, and consider them as concepts or attributes.
  - Use Cases are excellent description to draw for this analysis.
- Use a conceptual class category list
  - Make a list of candidate concepts.

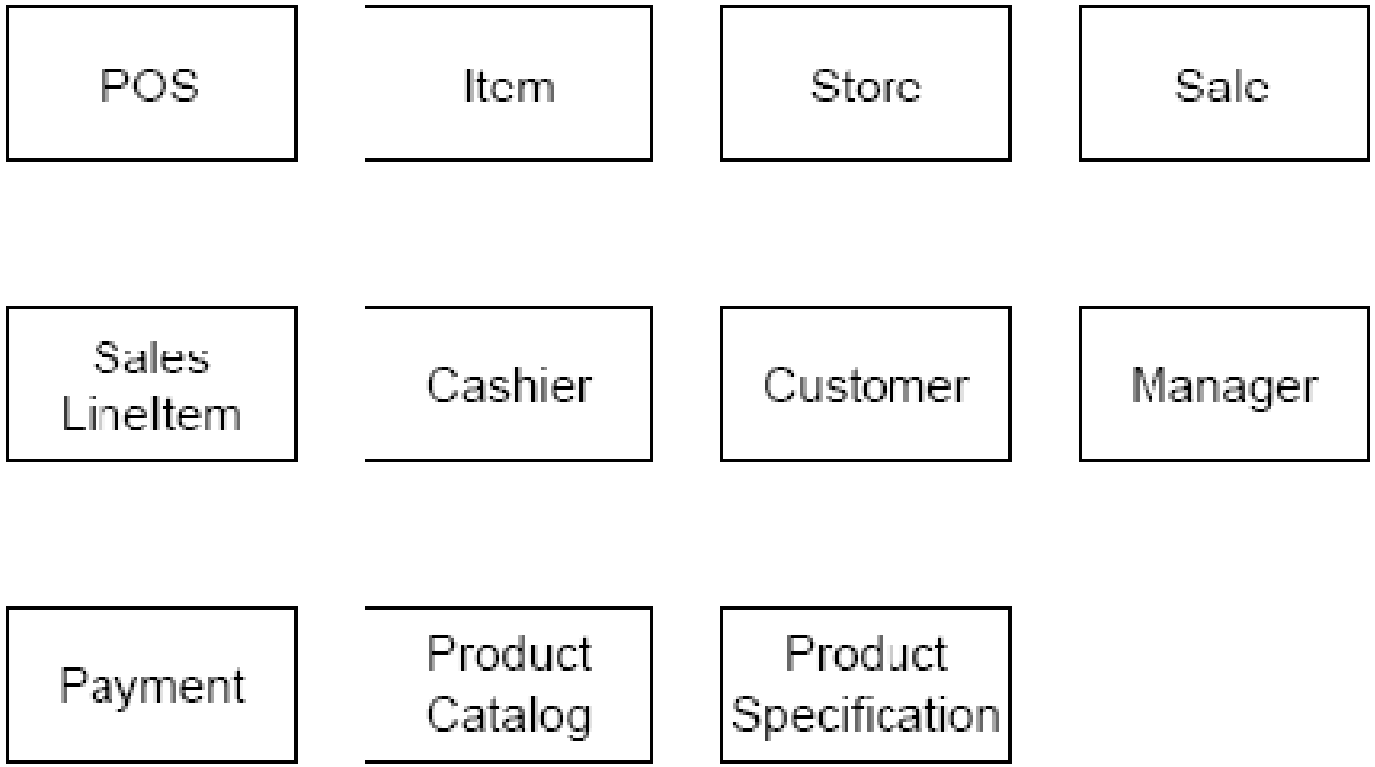
Finding Conceptual Classes with Noun Phrase Identification

1. This use case begins when a **Customer** arrives at a **POS checkout** with items to purchase.
2. The **Cashier** starts a new sale.
3. **Cashier** enters **item identifier**.

...

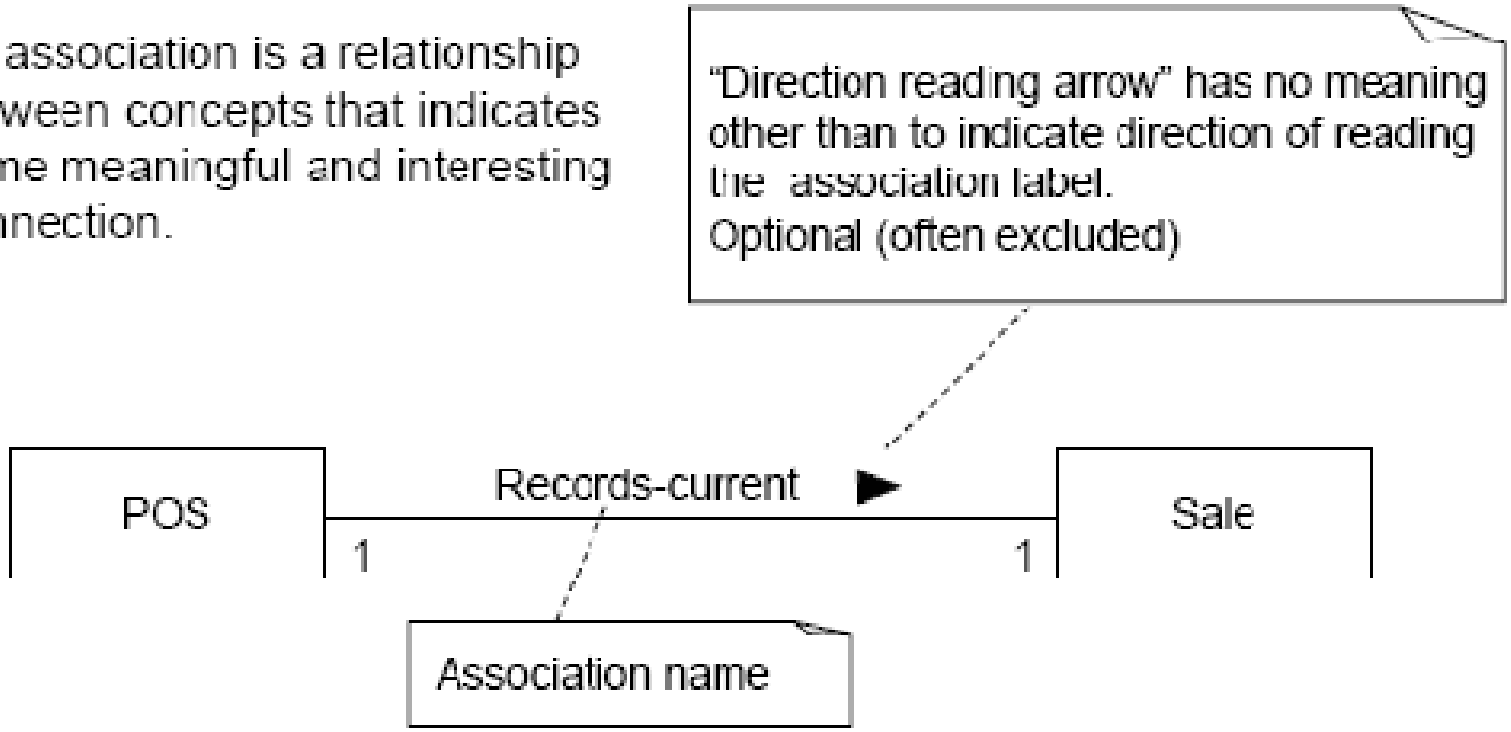
- The fully addressed Use Cases are an excellent description to draw for this analysis.
- Some of these noun phrases are candidate concepts; some may be attributes of concepts.
- A mechanical noun-to-concept mapping is not possible, as words in a natural language are (sometimes) ambiguous.

The NextGen POS (partial) Domain Model



Adding Associations

An association is a relationship between concepts that indicates some meaningful and interesting connection.

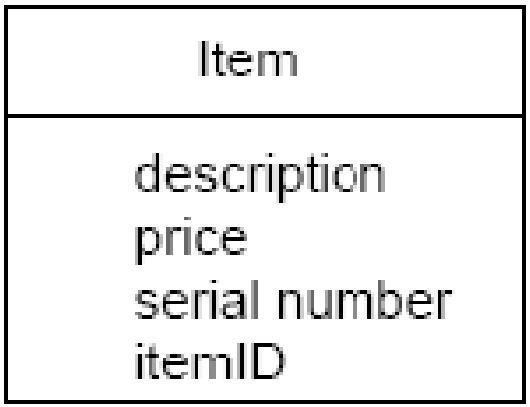


Use a conceptual class category list

Concept Category	Example
Physical or tangible objects	POS
Specifications, designs, or descriptions of things	ProductSpecification
Places	Store
Transactions	Sale, Payment
Transaction line items	SalesLineItem
Roles of people	Cashier
Containers of other things	Store, Bin

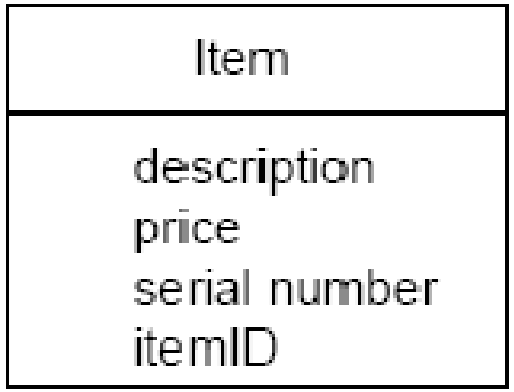
(See complete list in Larman 2nd. ed., pp. 134-135)

The Need for Specification or Description Conceptual Classes



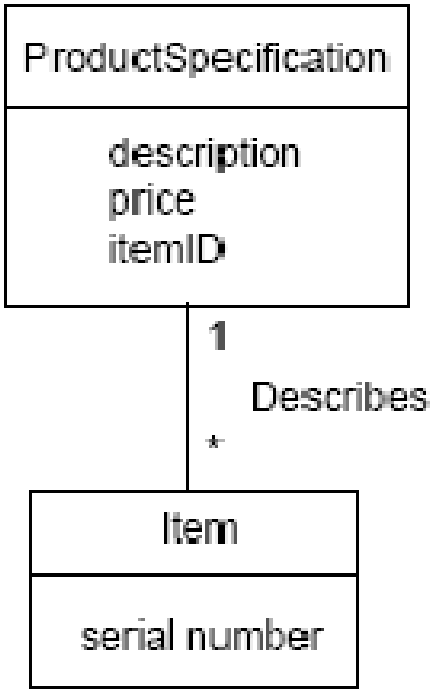
- What is wrong with this picture?
- Consider the case where all items are sold, and thus deleted from the computer memory.
- How much does an item cost?

The Need for Specification or Description Conceptual Classes



- The memory of the item's price was attached to inventoried instances, which were deleted.
- Notice also that in this model there is duplicated data (description, price, itemID).

The Need for Specification or Description Conceptual Classes

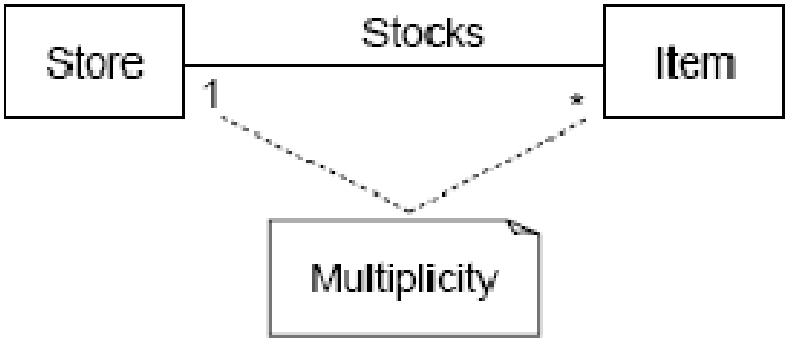


- Add a specification or description concept when:
  - Deleting instances of things they describe results in a loss of information that needs to be maintained, due to the incorrect association of information with the deleted thing.
  - It reduces redundant or duplicated information.

Finding Associations –Common Associations List

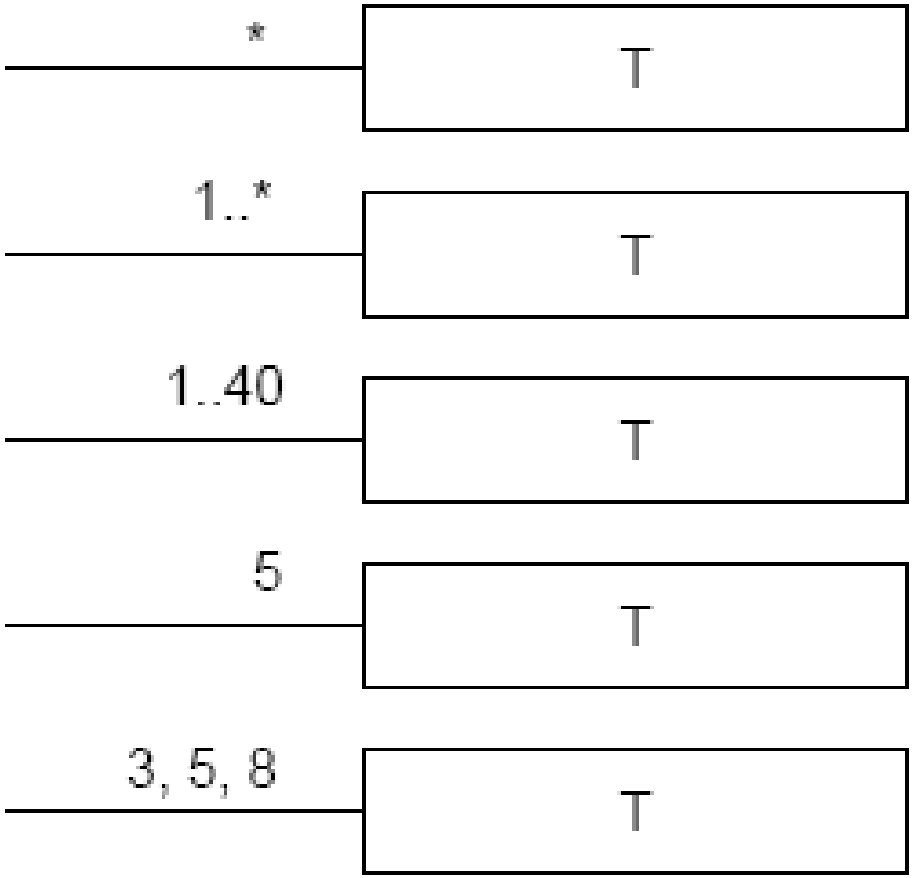
Category	Examples
A is a physical part of B*	Drawer - POS
A is a logical part of B	SalesLineItem - Sale
A is physically contained in/on B	POS Store
A is logically contained in B	ItemDescription - Catalog
A is a description of B	ItemDescription - Item
A is a line item of a transaction or report B	SalesLineItem - Sale
A is known/logged/recorded/ captured in B	Sale - POS
A is a member of B	Cashier - Store

Multiplicity



- Multiplicity defines how many instances of a type A can be associated with one instance of a type B, at a particular moment in time.
- For example, a single instance of a Store can be associated with "many" (zero or more) Item instances.

Multiplicity



Zero or more; "many"

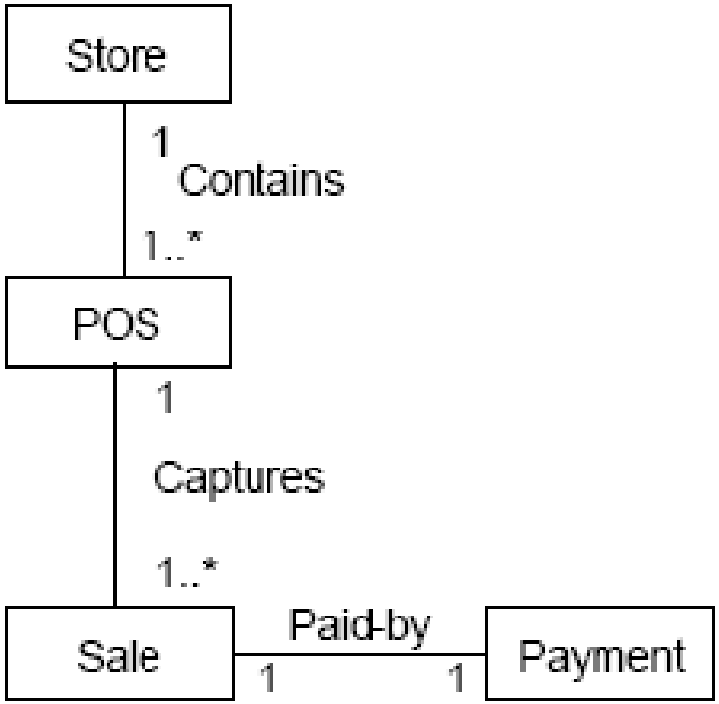
One or more

One to forty

Exactly five

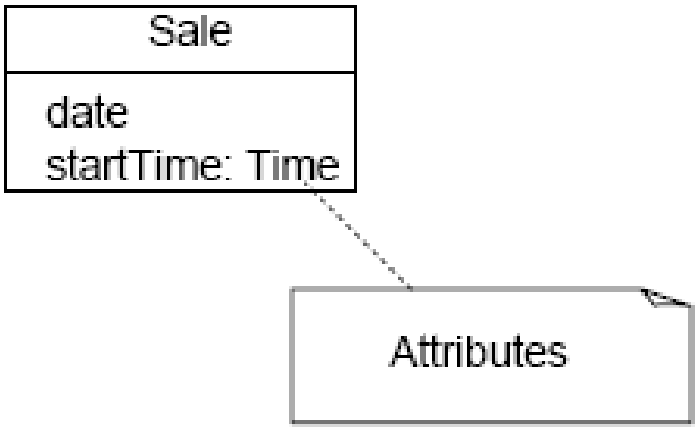
Exactly three, five or eight.

Naming Associations



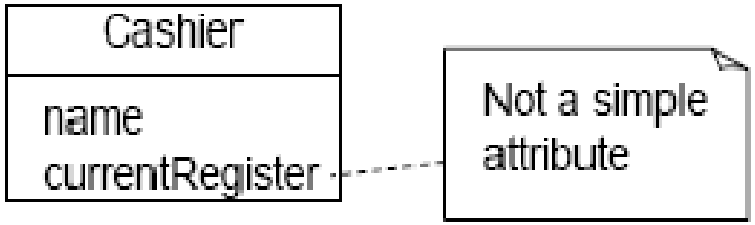
- Name an association based on a `TypeName-VerbPhrase-TypeName` format.
- Association names should start with a capital letter.
- A verb phrase should be constructed with hyphens.
- The default direction to read an association name is left to right, or top to bottom.

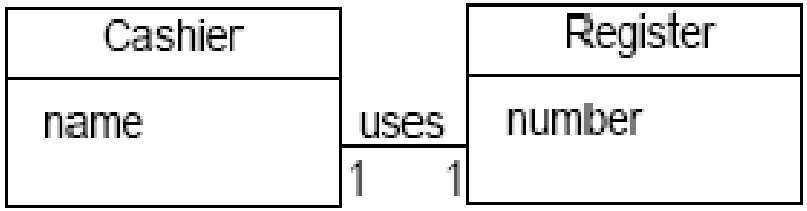
Adding Attributes



- An attribute is a logical data value of an object.
- Include the following attributes: those for which the requirements suggest or imply a need to remember information.
- For example, a Sales receipt normally includes a date and time.
- The Sale concept would need a date and time attribute.

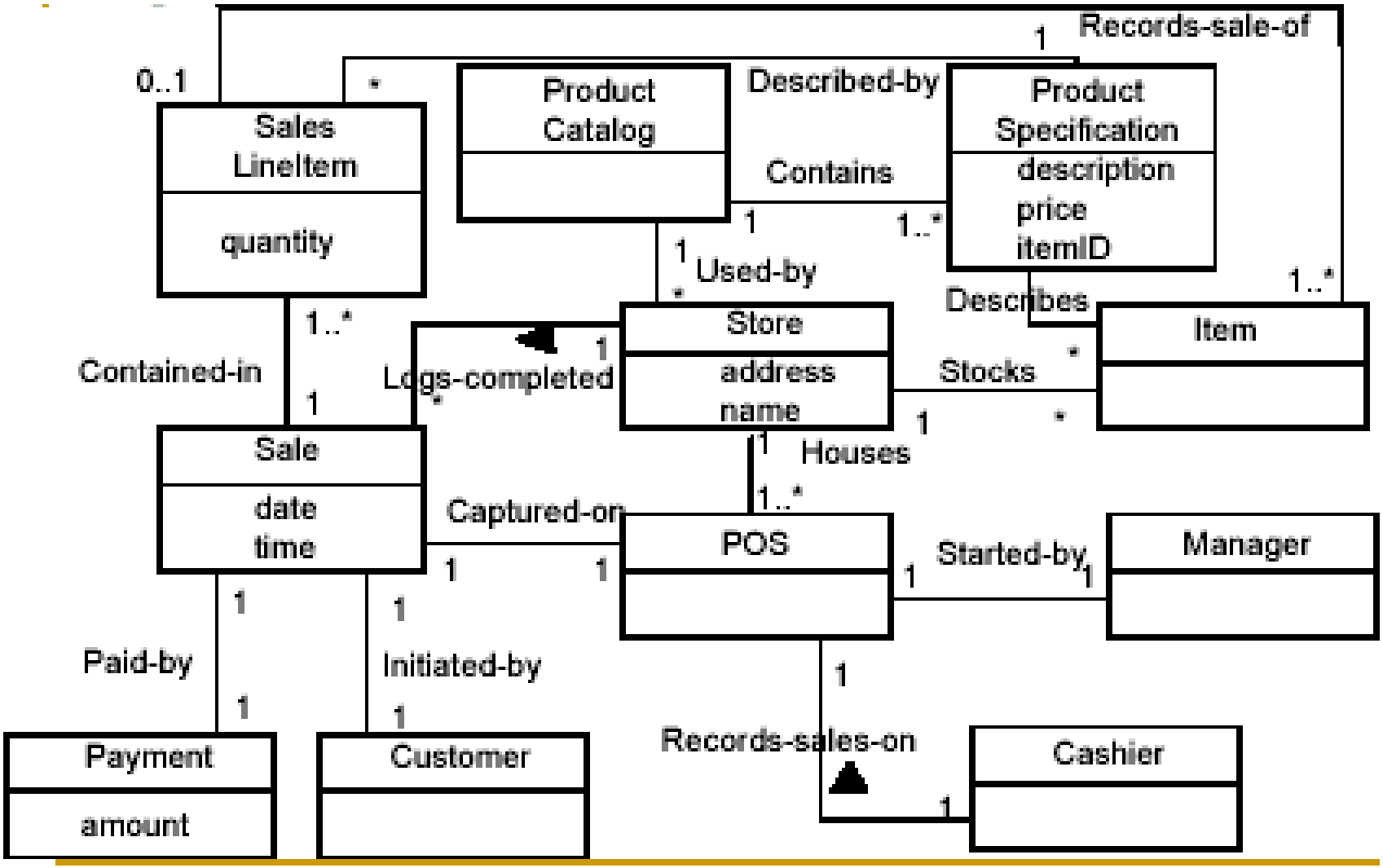
Valid Attribute Types





- Keep attributes simple.
- The type of an attribute should not normally be a complex domain concept, such as Sale or Airport.
- Attributes in a Domain Model should preferably be
  - Pure data values: Boolean, Date, Number, String, ...
  - Simple attributes: color, phone number, zip code, universal product code (UPC), ...

Domain Model Conclusion

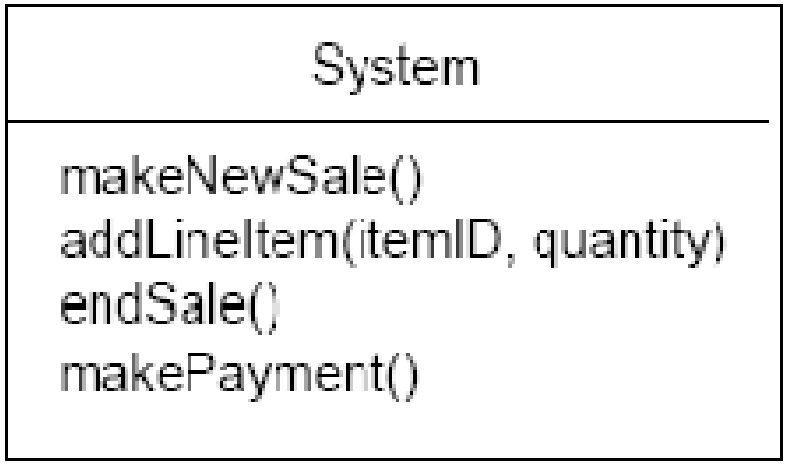


Use-Case Model: Adding Detail with Operation Contracts

Contracts

- Contracts are documents that describe system behavior.
- Contracts may be defined for **system operations**.
  - Operations that the system (as a black box) offers in its public interface to handle incoming system events.
- The entire set of system operations across all use cases, defines the public system interface.

System Operations and the System Interface



- In the UML the system as a whole can be represented as a class.
- Contracts are written for each system operation to describe its behavior.

Example Contract: addLineItem

Contract CO2: addLineItem

Operation: addLineItem (itemID: ItemID, quantity: integer)

Cross References: Use Cases: Process Sale.

Pre-conditions: There is a sale underway.

Post-conditions:

- A SalesLineItem instance *sli* was created. (instance creation)
- *sli* was associated with the Sale. (association formed)
- *sli.quantity* was set to quantity. (attribute modification)
- *sli* was associated with a ProductSpecification, based on itemID match (association formed)

Pre- and Postconditions

- Preconditions are assumptions about the state of the system before execution of the operation.
- A postcondition is an assumption that refers to the state of the system after completion of the operation.
  - The postconditions are not actions to be performed during the operation.
  - Describe changes in the state of the objects in the Domain Model (instances created, associations are being formed or broken, and attributes are changed)

addLineItem postconditions

- Instance Creation and Deletion
- After the itemID and quantity of an item have been entered by the cashier, what new objects should have been created?
  - A SalesLineItem instance *sli was created.*

addLineItem postconditions

- Attribute Modification
- After the itemID and quantity of an item have been entered by the cashier, what attributes of new or existing objects should have been modified?
- sli.quantity was set to quantity (attribute modification).

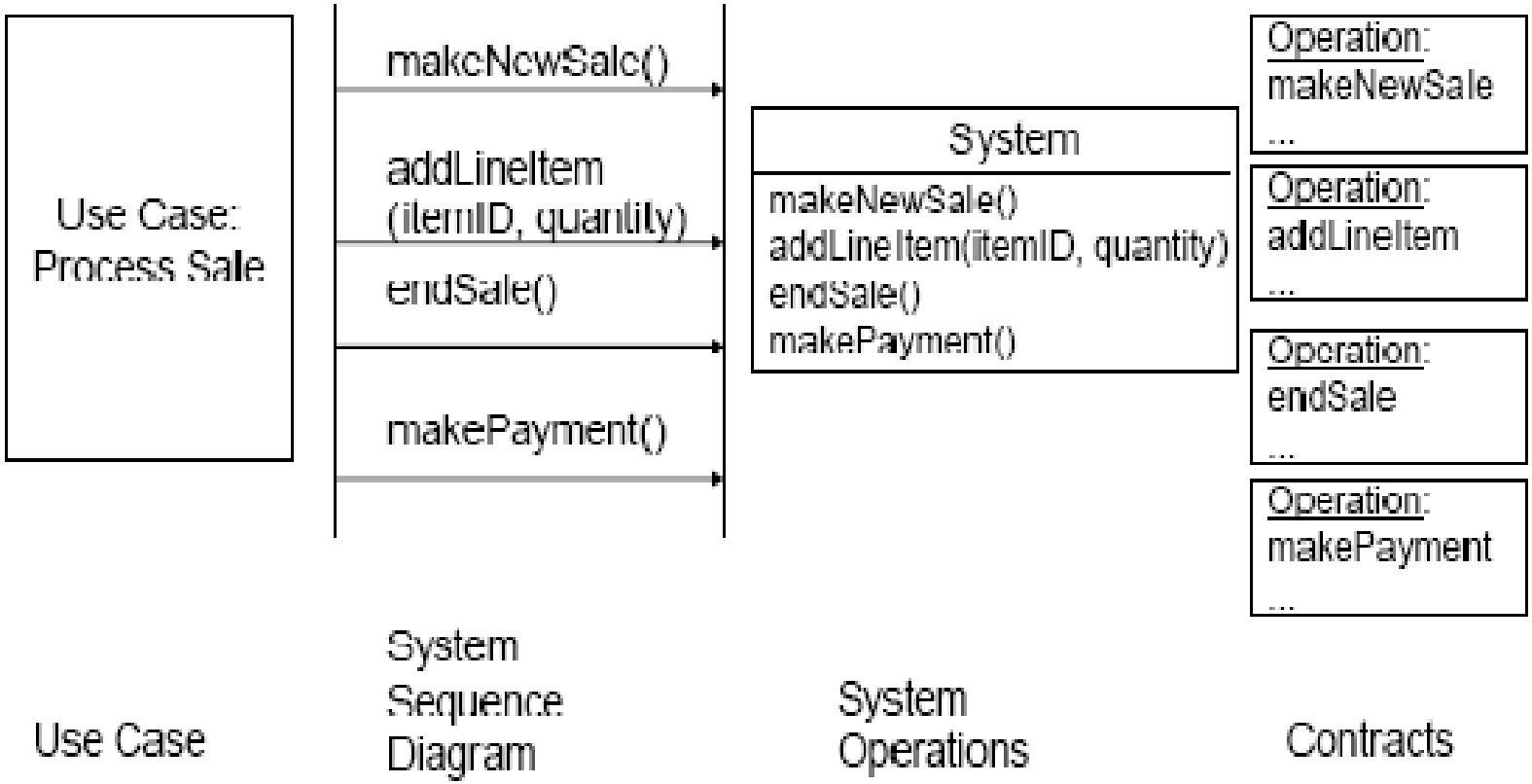
addLineItem postconditions

- Associations Formed and Broken
- After the itemID and quantity of an item have been entered by the cashier, what associations between new or existing objects should have been formed or broken?
  - sli was associated with the current Sale (association formed).
  - sli was associated with a ProductSpecification, based on itemID match (association formed).

Writing Contracts leads to Domain Model Updates

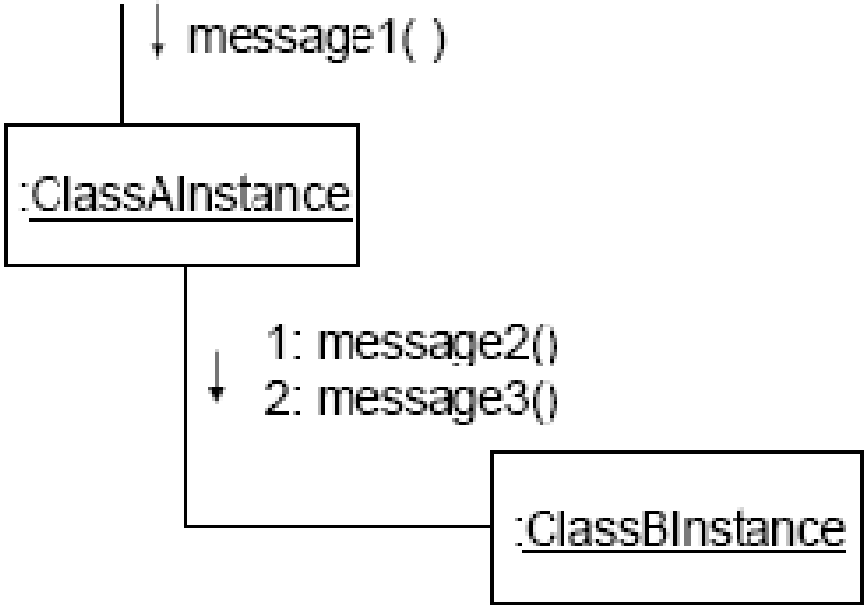
- It is also common to discover the need to record new concepts, attributes or associations in the Domain Model.

Guidelines for Contracts



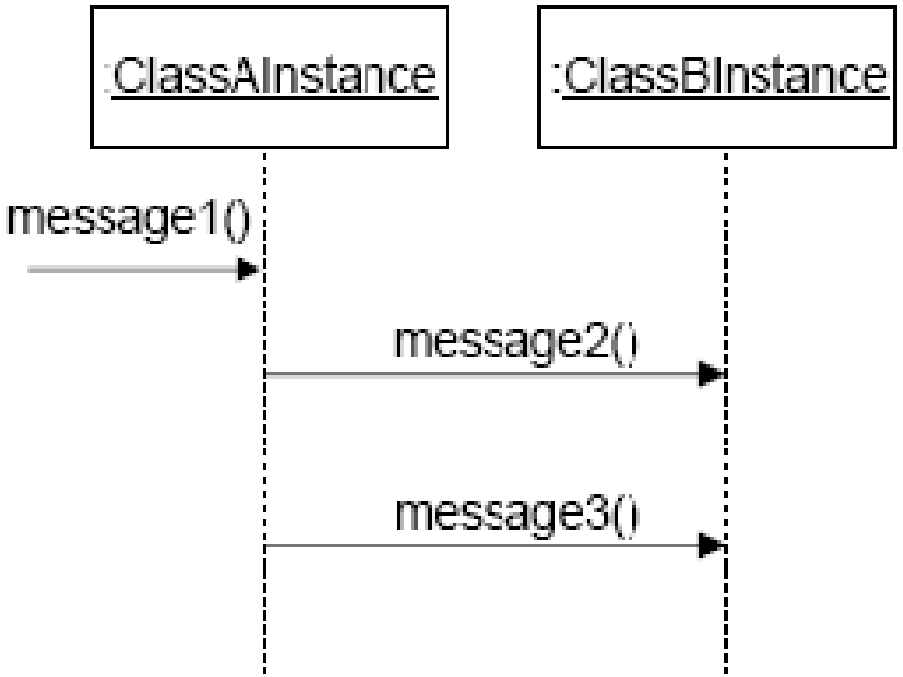
Interaction Diagram Notation

Introduction



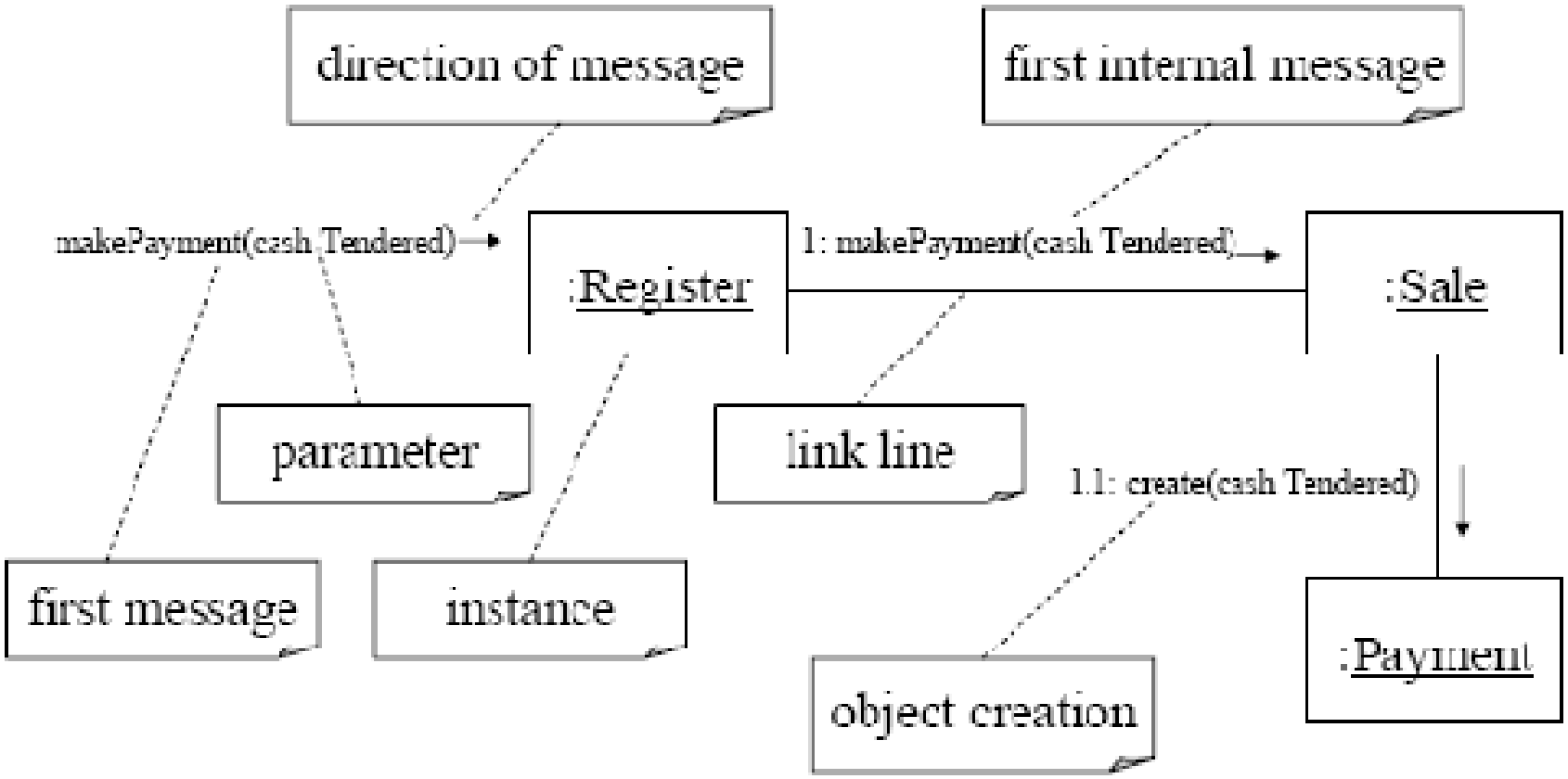
- *Interaction diagrams* illustrate how objects interact via messages.
- *Collaboration diagrams* illustrate object interactions in a graph or network format.

Introduction

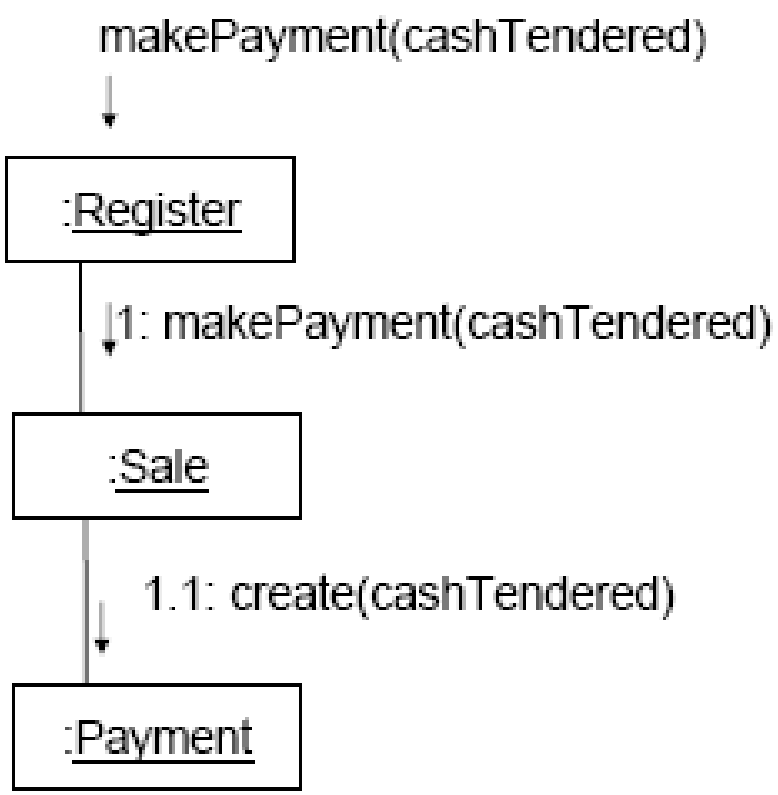


- *Sequence diagrams* illustrate interactions in a kind of fence format.
- Set of all operation contracts defines system behavior.
- We will create an interaction diagram for each operation contract.

Example Collaboration Diagram: makePayment

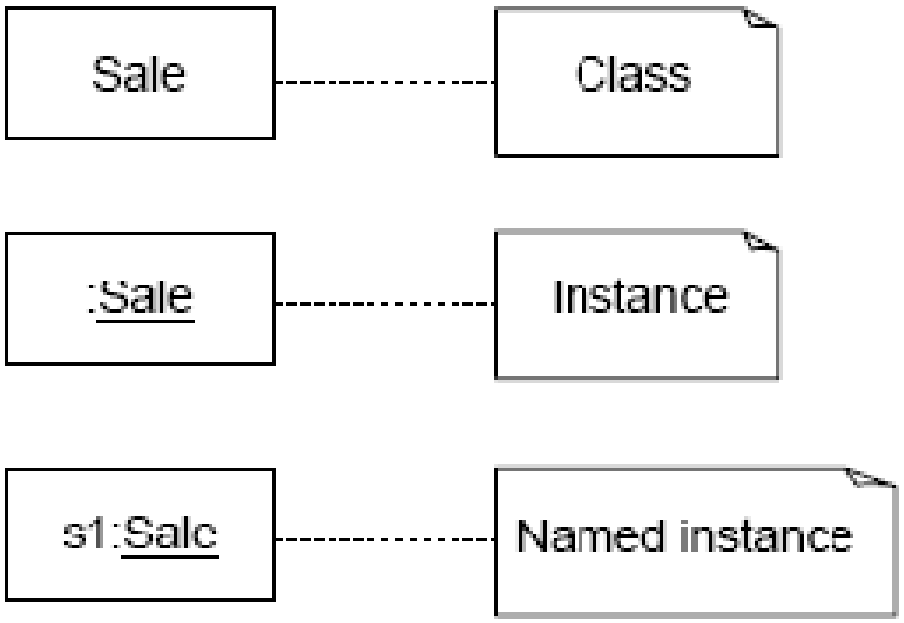


How to Read the makePayment Collaboration Diagram



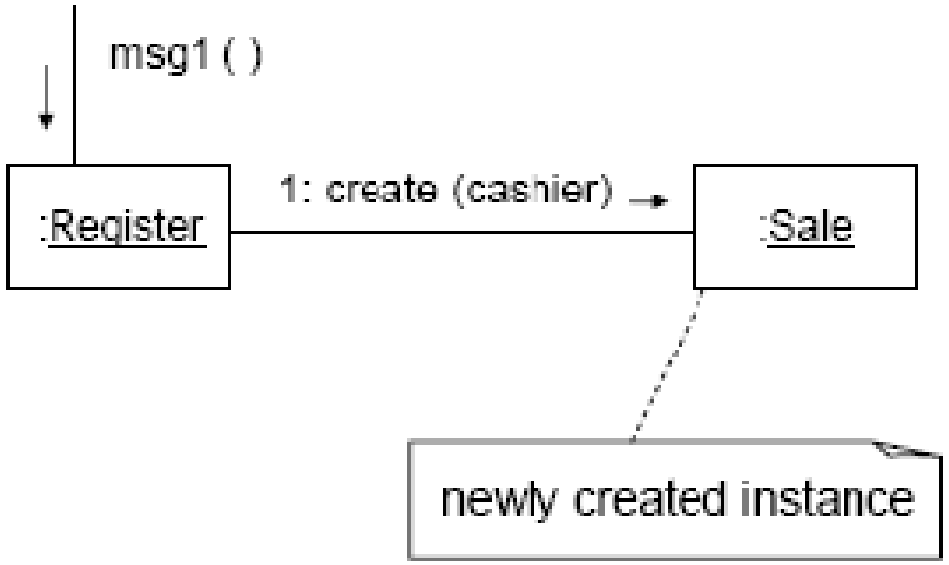
1. The message makePayment is sent to an instance of Register. The sender is not identified.
2. The Register instance sends the makePayment message to a Sale instance.
3. The Sale instance creates an instance of a Payment.

Illustrating Classes and Instances



- To show an instance of a class, the regular class box graphic symbol is used, but the name is underlined. Additionally a class name should be preceded by a colon.
- An instance name can be used to uniquely identify the instance.

Creation of Instances

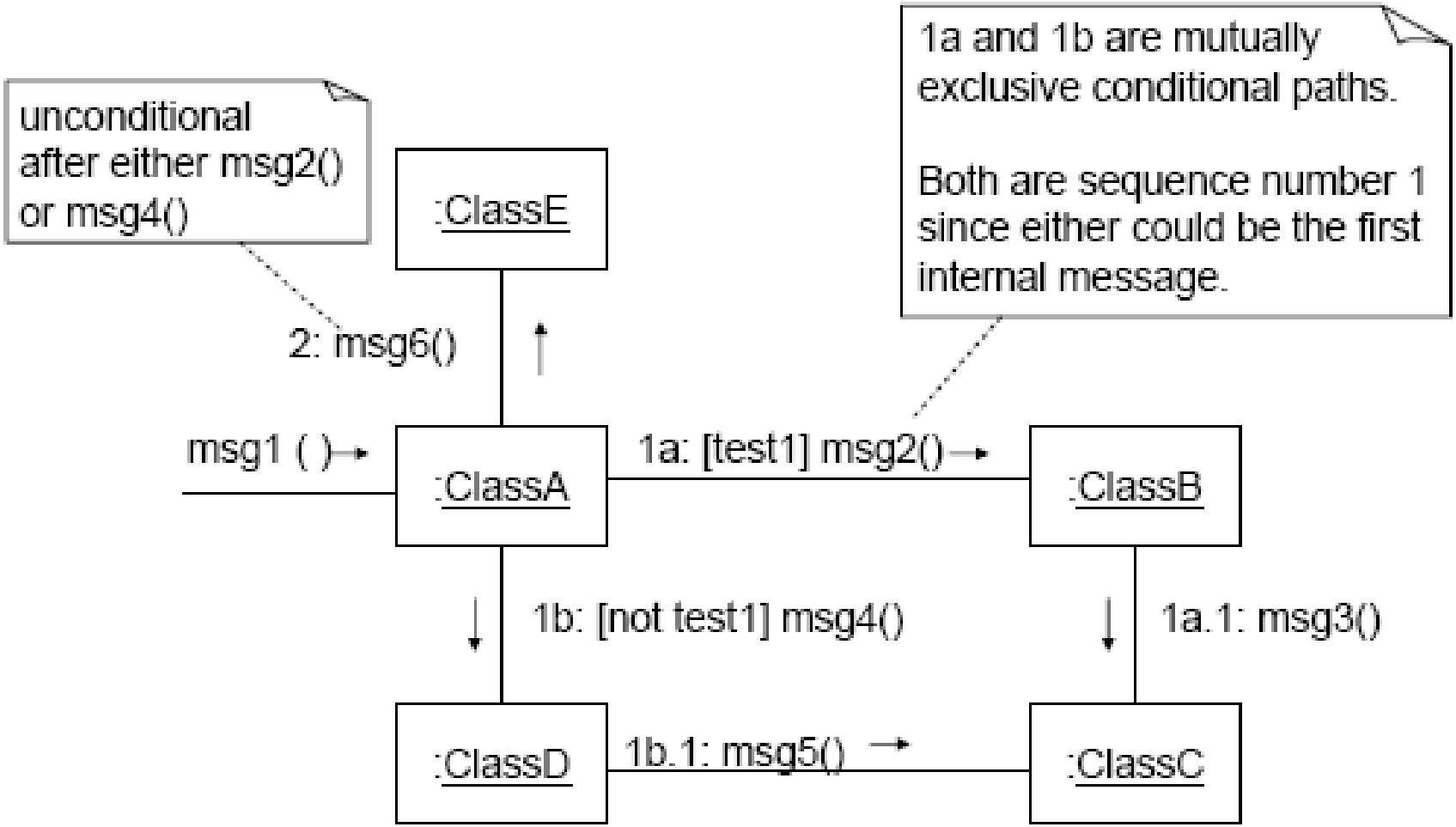


- The language independent creation message is create, being sent to the instance being created.
- The create message may include parameters, indicating passing of initial values.

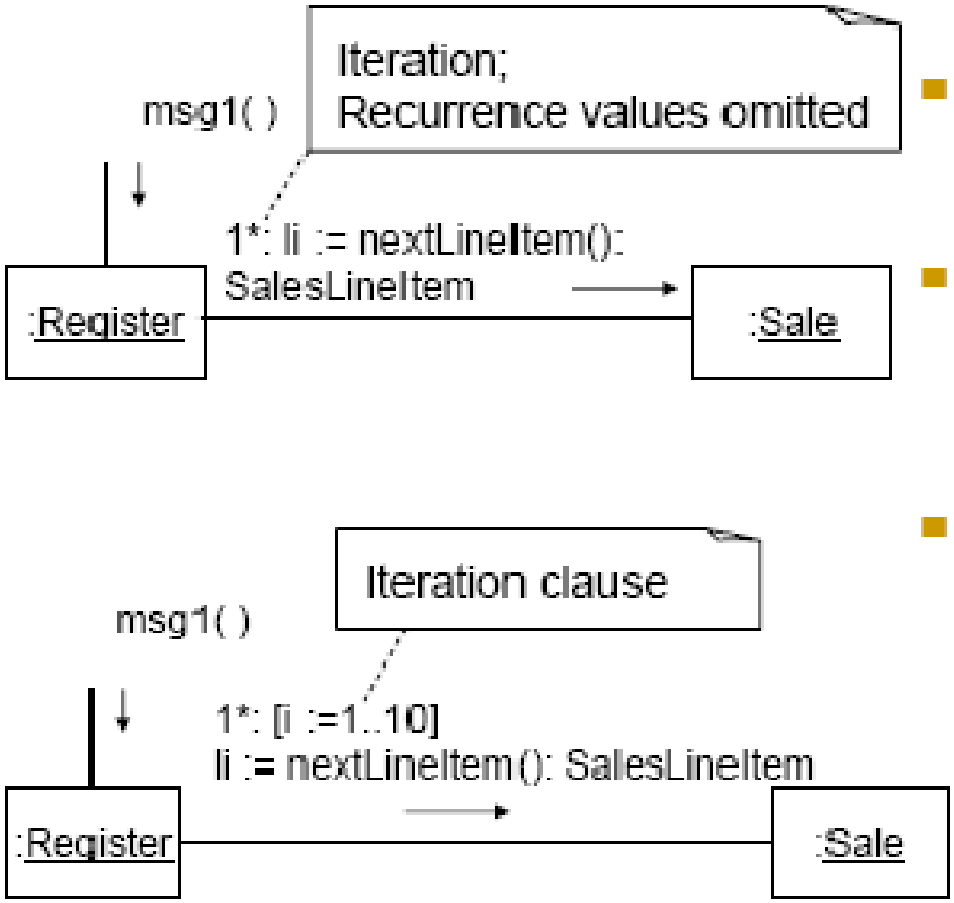
Conditional Messages

- A conditional message is shown by following a sequence number with a conditional clause in square brackets, similar to the iteration clause.
- The message is sent only if the clause evaluates to true.

Mutually Exclusive Conditional Paths



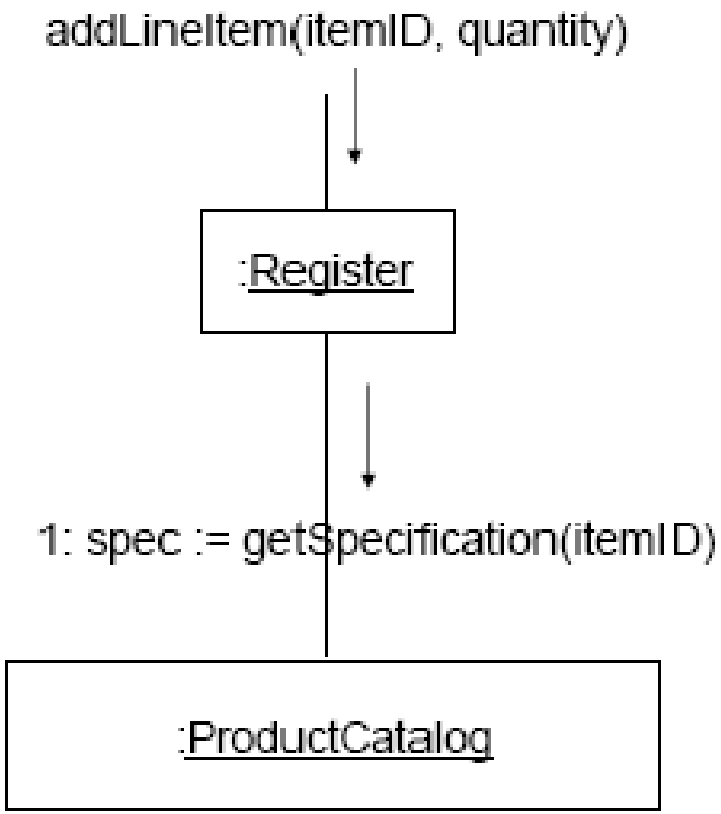
Iteration or Looping



- Iteration is indicated by following the sequence number with a star *
- This expresses that the message is being sent repeatedly, in a loop, to the receiver.
- It is also possible to include an iteration clause indicating the recurrence values.

Design Model: Determining Visibility

Visibility Between Objects

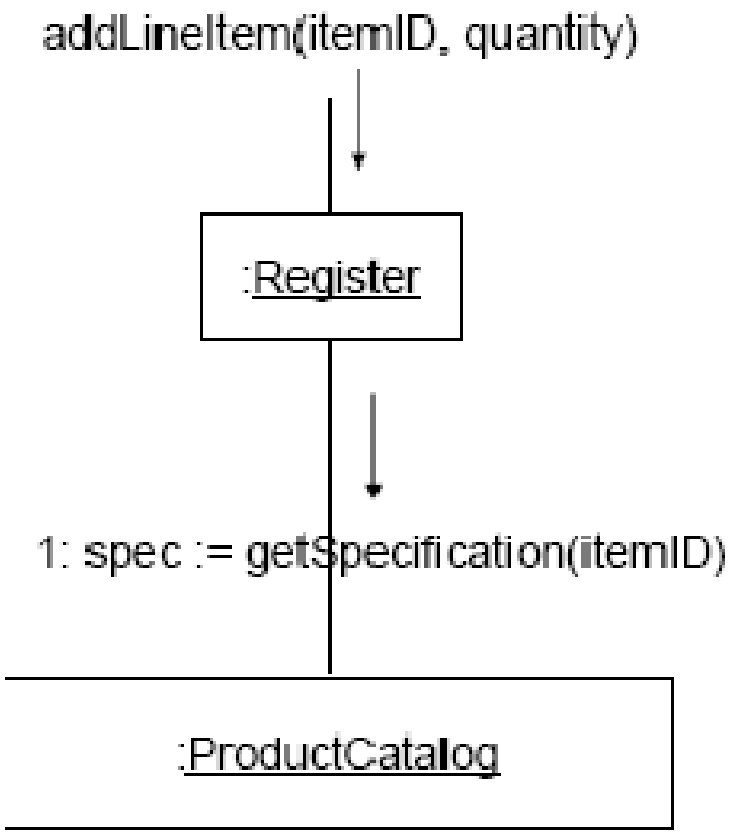


- The getSpecification message sent from a Register to a ProductCatalog, implies that the ProductCatalog instance is visible to the Register instance.

Visibility

- How do we determine whether one resource (such as an instance) is within the scope of another?
- Visibility can be achieved from object A to object B in four common ways:
  - Attribute visibility: B is an attribute of A.
  - Parameter visibility: B is a parameter of a method of A.
  - Local visibility: B is a (non-parameter) local object in a method of A.
  - Global visibility: B is in some way globally visible.

Visibility

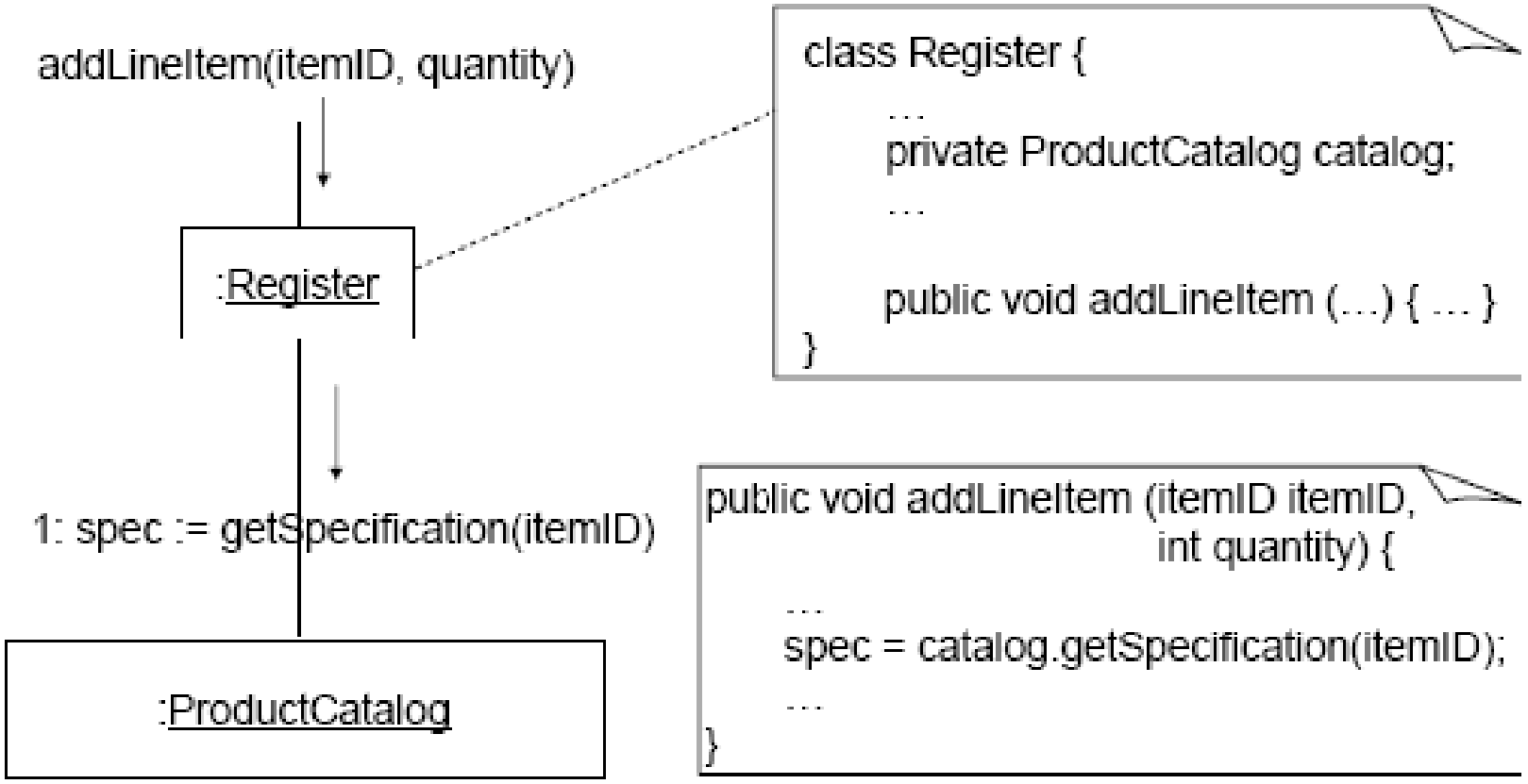


- The Regiser must have visibility to the ProductCatalog.
- A typical visibility solution is that a reference to the ProductCatalog instance is maintained as an attribute of the Register.

Attribute Visibility

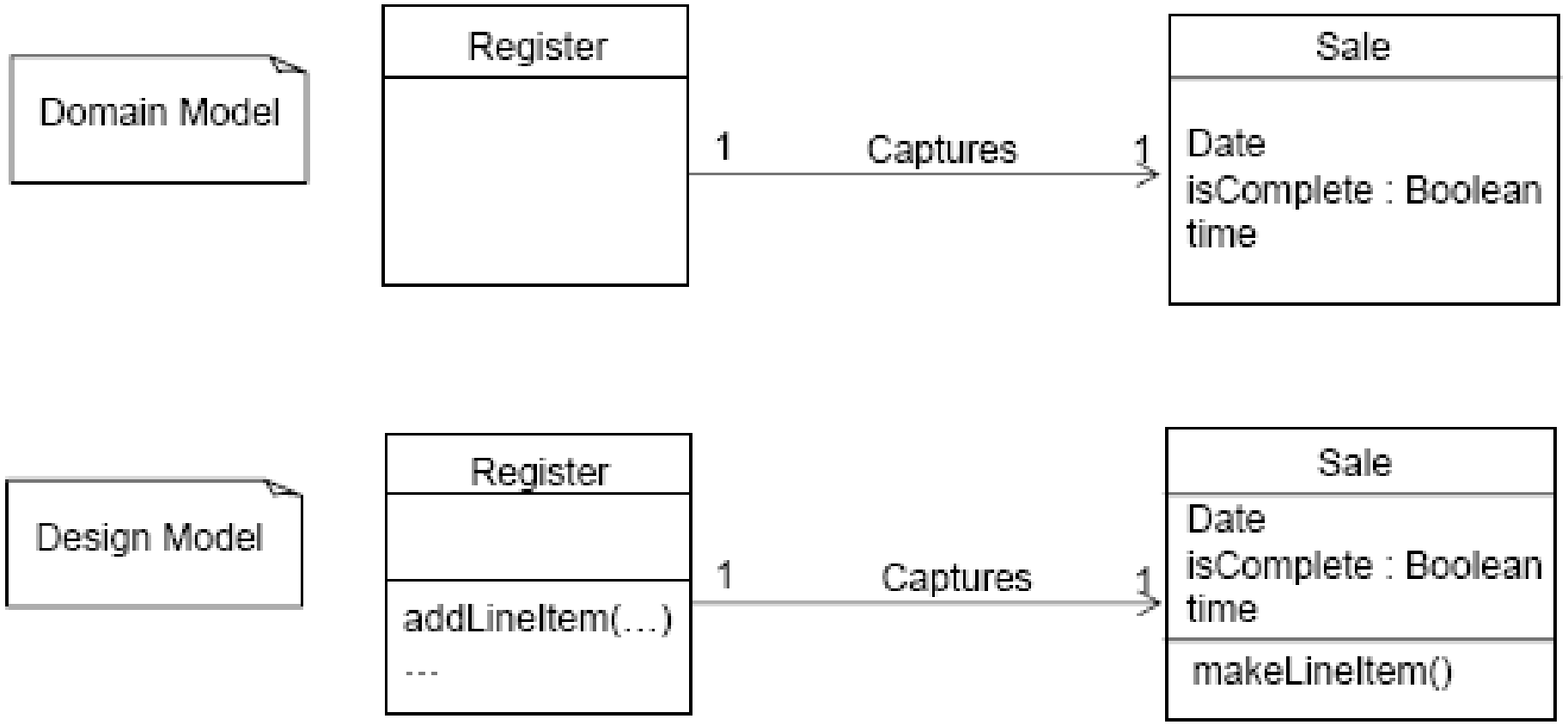
- Attribute visibility from A to B exists when B is an attribute of A.
- It is a relatively permanent visibility, becausenit persists as long as A and B exist.
- In the addLineItem collaboration diagram, Register needs to send message getSpecification message to a ProductCatalog. Thus, visibility from Register to ProductCatalog is required.

Attribute Visibility



Design Model: Creating Design Class Diagrams

Domain Model vs. Design Model Classes



Adding Navigability and Dependency Relationships