95-706

Midterm Exam Review

Richard J. Orgass
Management Information Systems Program
Carnegie Mellon University

Carnegie Mellon

Agenda

- Steps in analysis and design
- Identifying Classes for Models
- Use Cases
  - diagrams
  - textual
- Class Diagrams
- Links and Associations
- Contracts
- Interaction Diagrams
  - Sequence Diagrams
  - Collaboration Diagrams
- Messages

One Set of Steps Analysis

- If you don't understand the problem area at all:
  - Mark all the nouns and noun phrases in the problem description
  - List the nouns and noun phrases -- candidates for classes
  - Prune this list using the narrative text and domain knowledge
- Working with the customer
  - construct use cases, textual and graphical
  - continue iterating until both you and the customer agree that the use cases cover what is needed
  - add contracts to the use cases
  - (optional) validate use cases by comparing the nouns and noun phrases they contain to the lists of noun phrases prude as above.
  - Collect the textual uses cases and draw use case diagrams to guide readers
  - the use cases become the functional specifications in the software development contract

One Set of Steps -- II
Analysis

- Class diagrams
  - Create class diagrams from the use cases
  - Add attributes and operations to the classes
  - (optional) Compare the classes obtained from use cases to the ones from the noun phrases. Update as appropriate
  - Look for generalizations in your diagrams
    - Order
      - business order
      - personal order
    - Trade
      - sale trade
      - purchase trade
  - Add associations to the diagrams
  - Add multiplicities (constraints) to the diagrams
  - Add contracts for the methods (from use cases)
  - Review class diagrams with customer if possible.

One Set of Steps -- III
Design

- Sequence Diagrams (using objects)
  - Create them if you need them
  - Critical timing specifications require sequence diagrams
  - Uses
    - Longest delay analysis
    - Deadlock analysis for concurrent processes
    - Help model data flow
    - Completeness check for Collaboration Diagrams
- Collaboration Diagrams (using objects)
  - Messages from object to object determine contracts for methods
  - Adding a defined message means
    - target object performs the operation
      - directly
      - using services of other objects
  - Assigning Responsibilities for operations specified by use cases is the principal activity when creating collaboration diagrams

One Set of Steps -- IV
Design

- Message Assignment Completed
  - use UML tool to generate class declarations
  - fill in method bodies using the contracts associated with the messages received and sent by the object
  - compile, link and test
- Test Cases
  - Generated from the use cases
  - There must be a test case for every path in the collaboration diagrams
  - All use cases must be accounted for in the collaboration diagrams
    - testing team confirms that this is the case

Identifying Classes

for

Models

First Understanding of Problem

Mark Nouns and Noun Phrases

The **Portfolio Manager** shall be able to roll up **portfolios** on several levels.

A **Trader** shall be able to place orders, on behalf of a portfolio, that generates one or more trades.

A **Portfolio Manager** shall be able to select a **pair-off method** in conjunction with placing a **sell order**.

The **entry** of a **trade** shall generate **forecasted cash-flows** associated with the given **trade lot**.

The **system** shall match up **actual cash-flows** with **forecasted cash-flows**.

The **system** shall automatically generate appropriate **postings** to the **General Ledger**.

The **system** shall allow an **Assistant Trader** to modify trade data and propagate the **results** appropriately.

First Understanding of Problem

First Set of Candidate Classes

actualcashflow	order	sell order
Assistant Trader	pairoff method	system
entry	portfolio	trade
forecasted cashflow	Portfolio Manager	trade data
General Ledger	posting	trade lot
level	results	Trader

Refined Set of Candidate Classes

ActualCashFlow	PairoffMethod	SellOrder
BuyOrder	Portfolio	Trade
ForecastedCashFlow	Posting	TradeLot

Use Cases

Simple Use Case Example

- Point-of-Sale System
  - computerized system used to record sales and handle payments
  - typically used in a retail store
  - hardware and software components
    - computer
    - bar code scanner
    - software to run the system
- Problem
  - create software to run a point-of-sale terminal
  - using an interactive-incremental development strategy
    - requirements
    - object oriented analysis
    - design
    - implementation

Simple Use Case Example
Textual Use Case Heading

Use Case:	Buy Items with Cash
Actors:	Customer (initiator), Cashier
Purpose:	Capture a sale & it's cash payment
Overview:	A customer arrives at a checkout with items to purchase. The cashier records the purchase items and collects payment. On completion, the customer leaves with the items.
Type:	Primary and essential.
Cross Refs:	R1.1, R1.2, R1.3, R1.7, R1.9, R2.1, R2.2, R2.3, R2.4

Typical Sequence of Events

Actor Action	System Response
1. Customer arrives at a POST checkout with items to purchase.	
2. Cashier records the identifier from each item. If there is more than one of the same item, the Cashier can enter the quantity as well.	3. Determines the item price and adds the item information to the running sales transaction. The description and price of the current item are presented.
4. On completion of item entry the Cashier indicates to the POST that item entry is complete.	5. Calculates and presents the sales total.

Typical Sequence of Events - 2

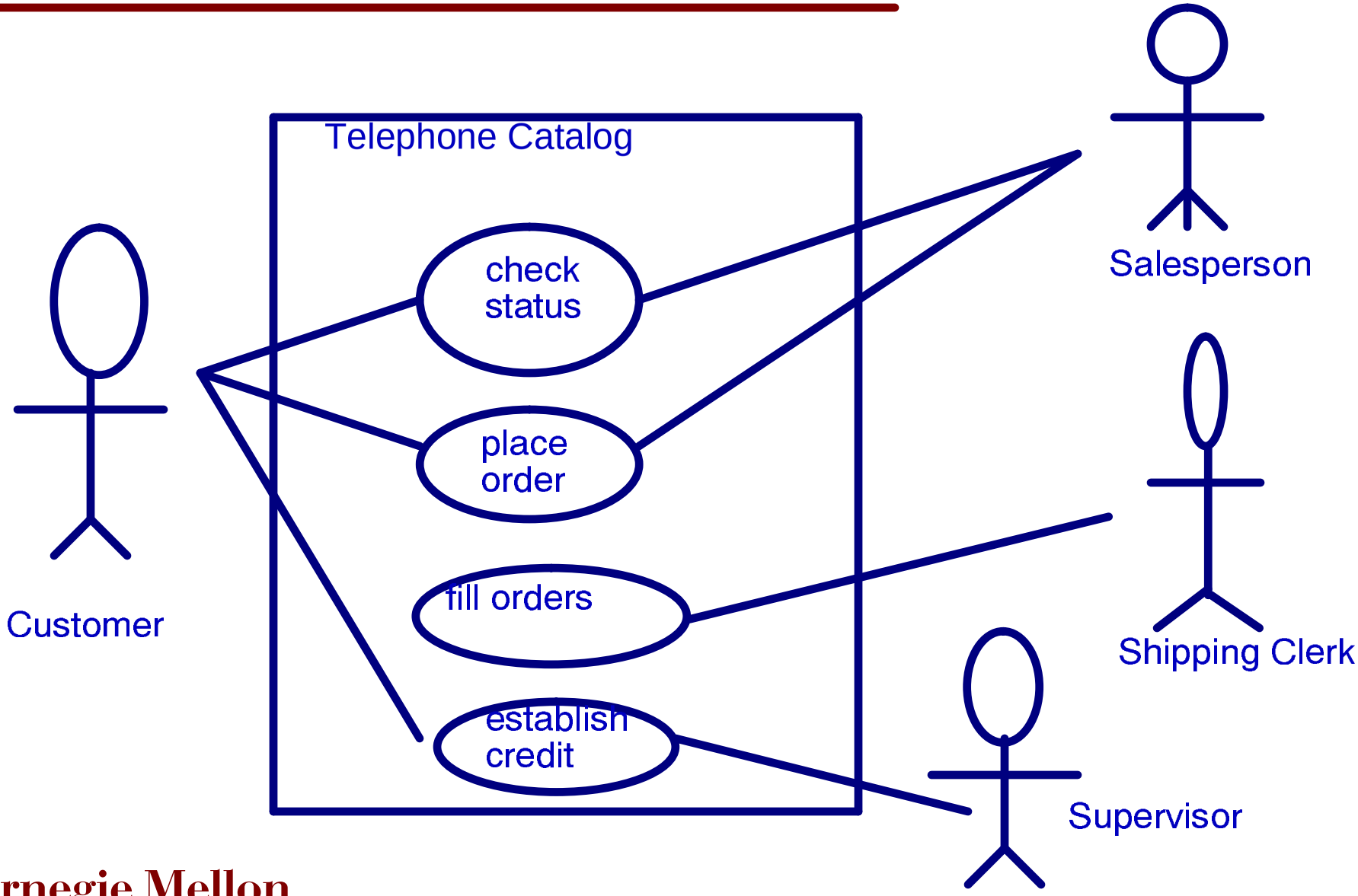
Actor Action	System Response
6. Cashier tells Customer the total.	
7. Customer gives a cash payment -- "cash tendered" -- possibly greater than the sale total.	
8. Cashier Records the cash received amount	
	9. Shows the balance due back to the Customer
10. Cashier deposits the cash received and extracts the balance owing Cashier gives the balance owing and the printed receipt to customer.	
	11. Logs the completed sale.

Typical Sequence of Events -- 3

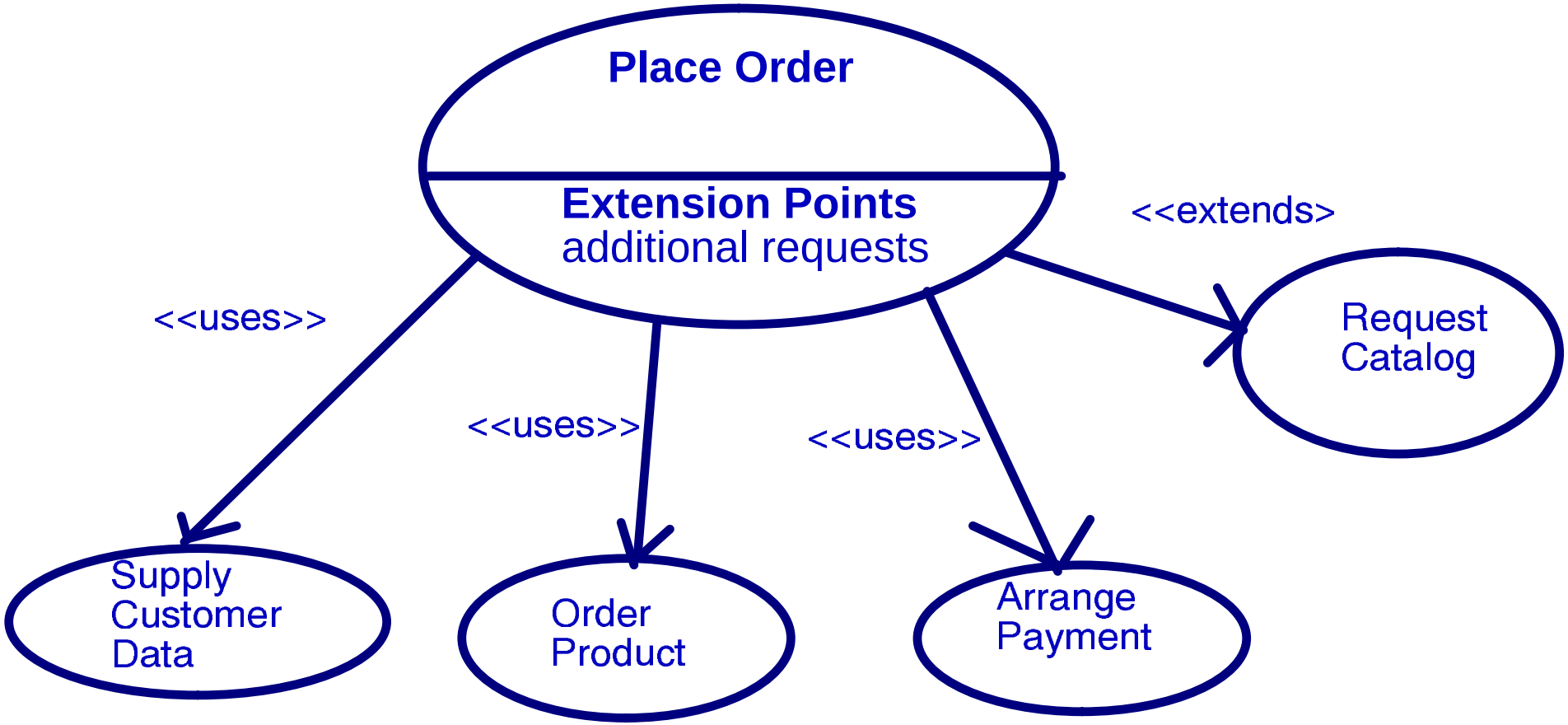
Final Actor Action

12. The Customer leaves with the items purchased

Example Use Case Diagram



Example Use Case Diagram - 2



Common Mistake with Use Cases

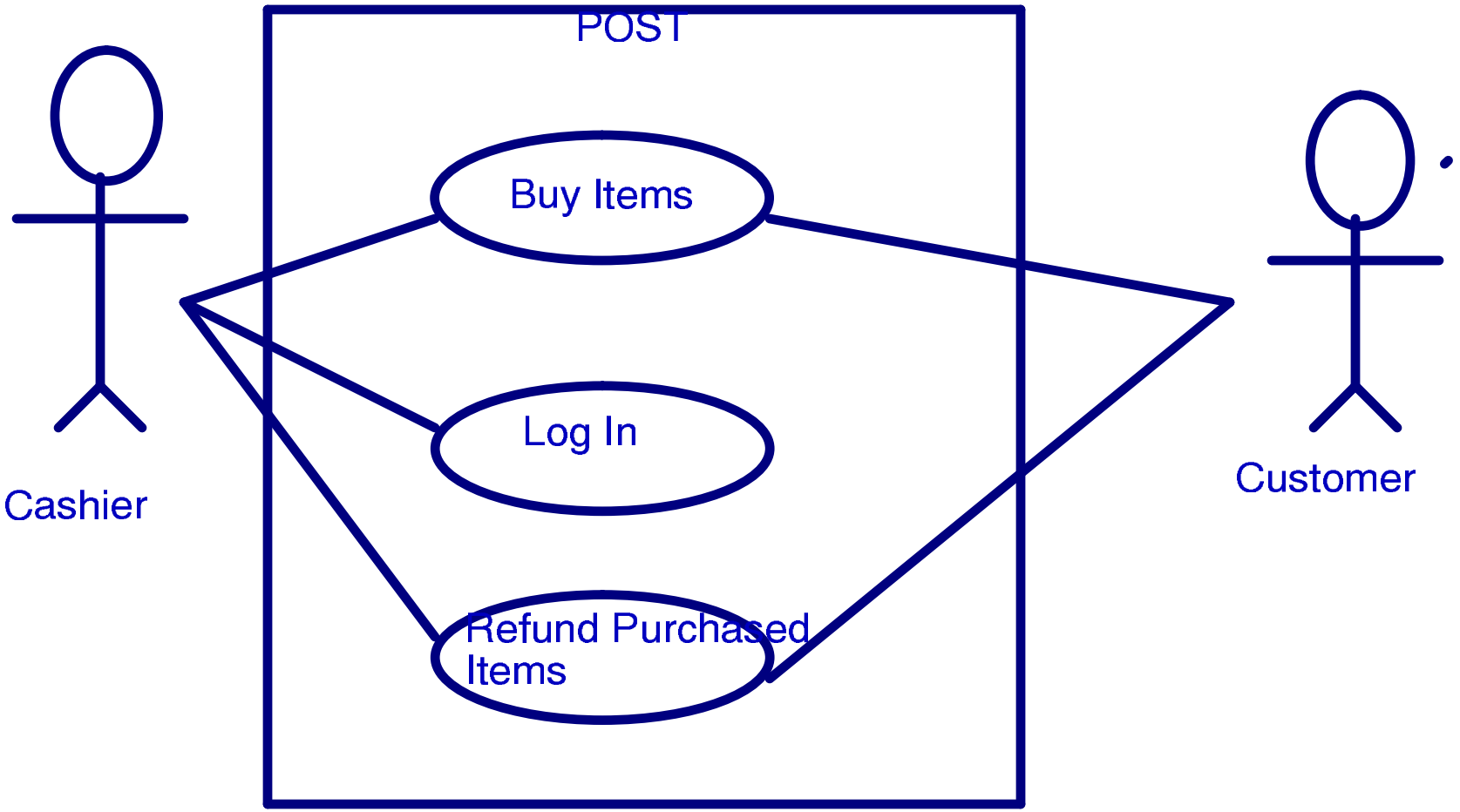
- Common Use Case Identification:
  - represent individual steps, operations or transactions as use cases
  - In POST system, "Printing the Receipt" as a Use Case instead of part of use case process "Buy Items"
- Can break down activities or portions of a use case into sub-use cases (called abstract use cases) -- even down to one step -- but this is not the norm. Might discuss later.

C

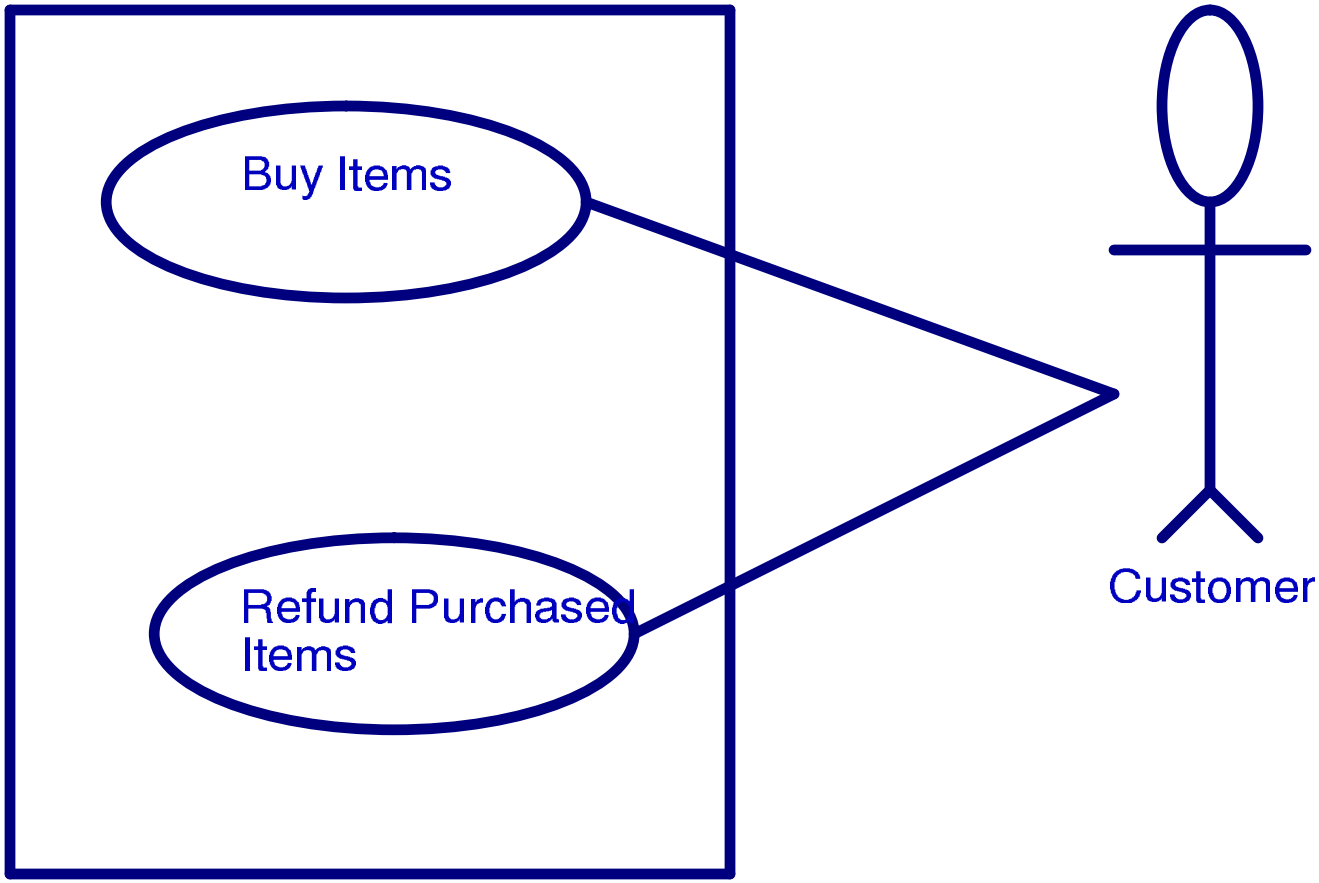
Identifying Use Cases

- Actor Based
  - Identify the actors related to a system or organization
  - For each actor, identify the processes they initiate or participate in.
- Event Based
  - Identify the external events that a system must respond to
  - Relate the events to actors and use cases
- Example: Point of Sale Application
  - Actors
    - Cashier, Customer
  - Events
    - Cashier initiated
      - Log In
      - Log Out
    - Customer
      - Buy Items
      - Refund Items

Use Case: POST is System



Use Case: Store is System



Essential vs. Real Use Cases

- Essential Use Cases
  - **Essential use cases are expanded use cases that are expressed in an ideal form that remains relatively free of technology and implementation details; design decisions are deferred and abstracted, specially those related to the user interface. An essential use case describes the process in terms of its essential activities and motivation. The degree of abstraction by which one is described exists on a continuum; use cases may be more or less essential in their description.**
  - **High-level use cases are always essential in nature, due to their brevity and abstraction.**
    - Constantine, L. The Case for Essential Use Cases. "Object Magazine", May 1997. Ny. NY: SIGS Publications

Essential and Real Use Case (ATM)

Actor Action (essential)
1. The Customer identifies themselves.
3. and so on

System Response (essential)
2. Presents Options
4. and so on

Actor Action (real)
1. Customer inserts their card.
3. Enters PIN on keypad.
5. and so on

System Response (real)
2. Prompts for PIN
4. Displays Options menu
6. and so on.

Essential Buy Items Use Case

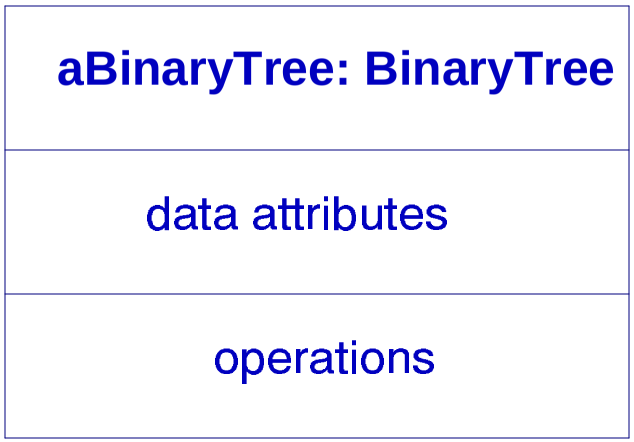
Actor Action	System Response
1. Cashier records the identifier from each item	2. Determines the item price and adds the item information to the running sales transaction
If there is more than one of the same item, the cashier can enter the quantity as well.	The description and price of the current item are presented.
3. and so on.	4. and so on

Real Buy Items Use Case

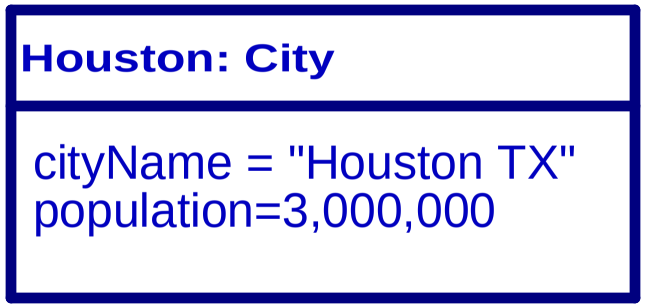
Actor Action	System Response
1. For each item, the Cashier types in the Universal Product Code (UPC) in the UPC input field of Window1. They then press the "Enter Item" button with the mouse or by pressing the <Enter> key.	2. Displays the item price and adds the item information to the running sales transaction.
3. etc.	The description and price of the current item are displayed in Textbox2 of Window1. 4. etc.

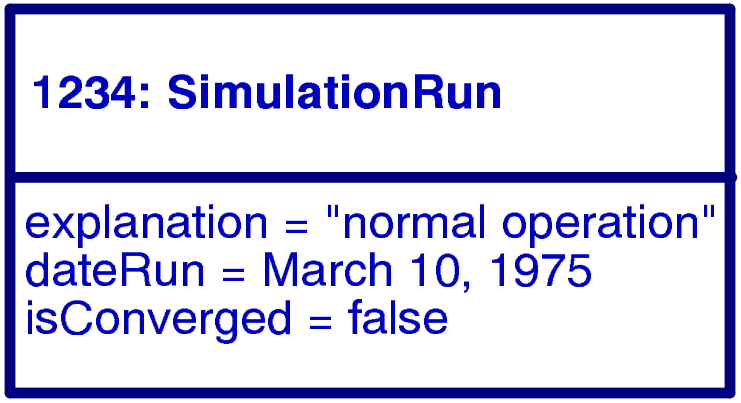
Class Diagrams

Object Diagrams

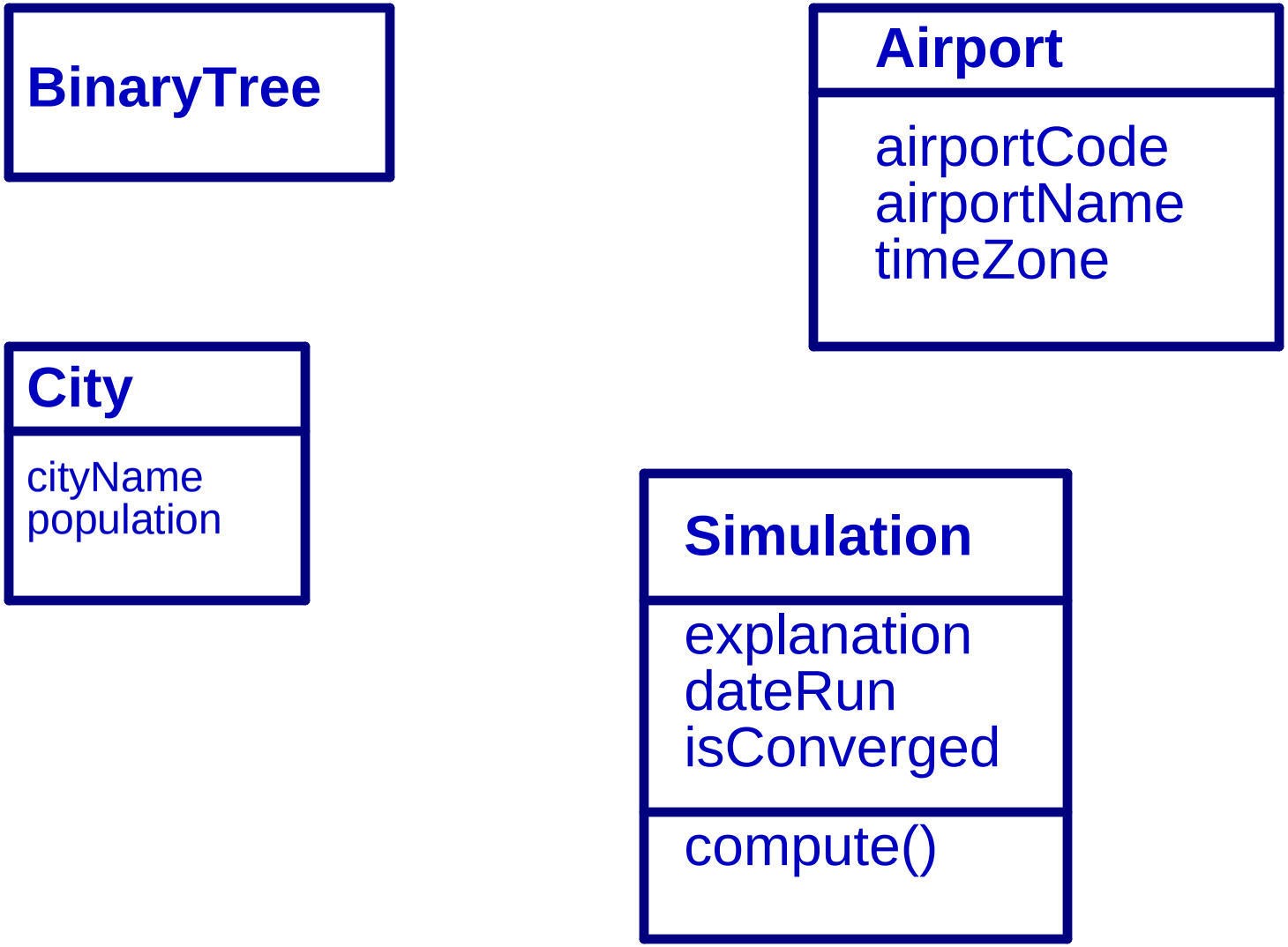


Examples of Objects

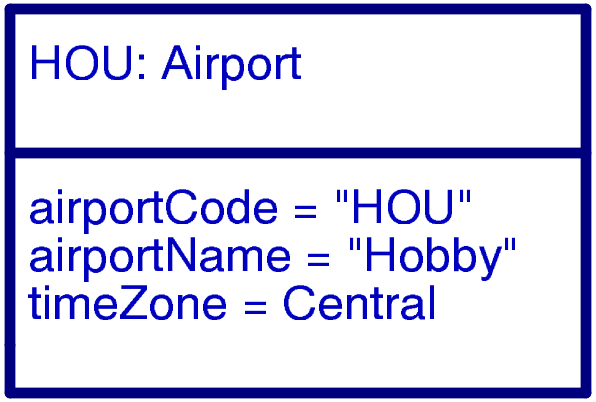




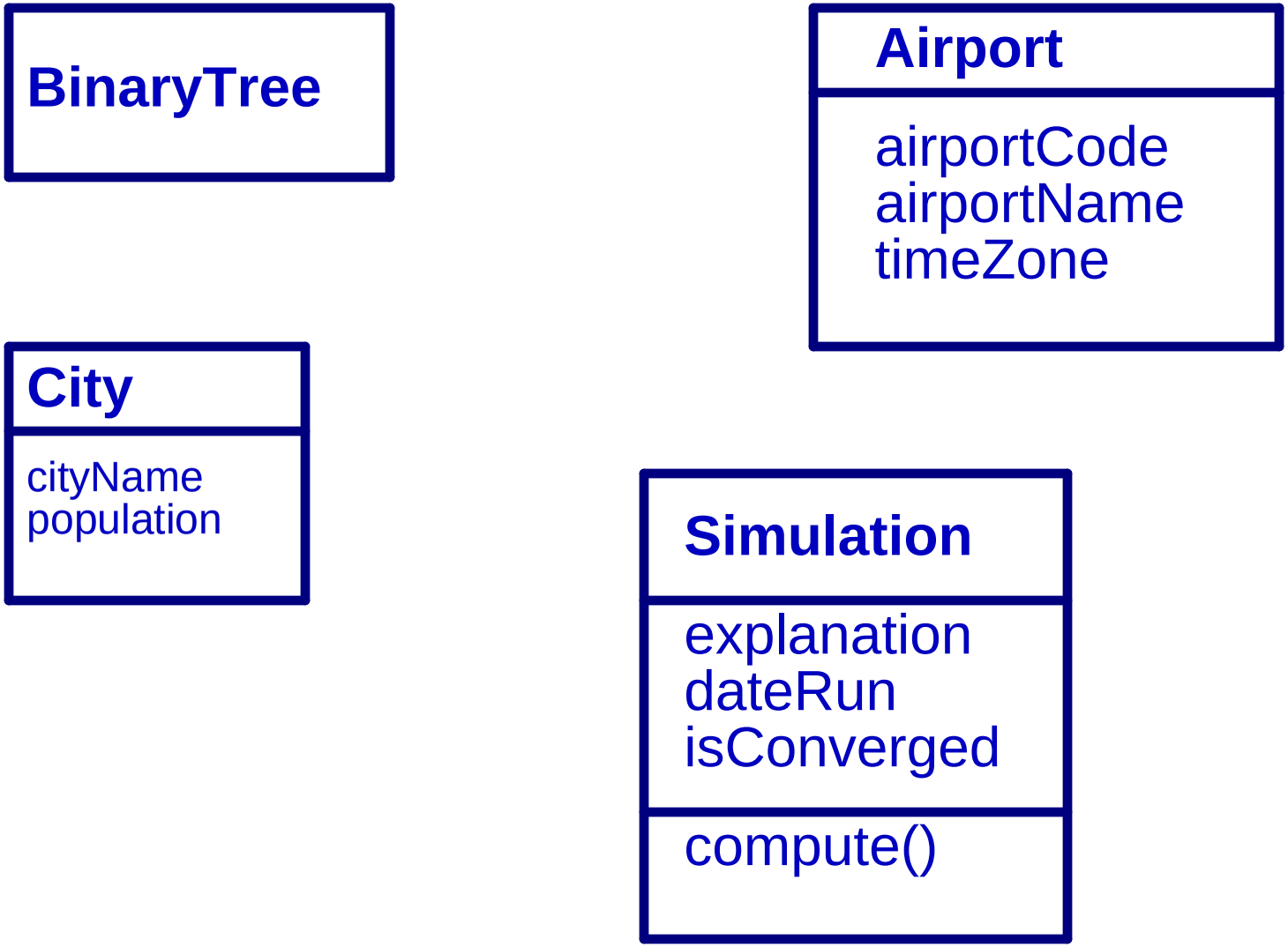
Class Diagrams



More Examples of Objects

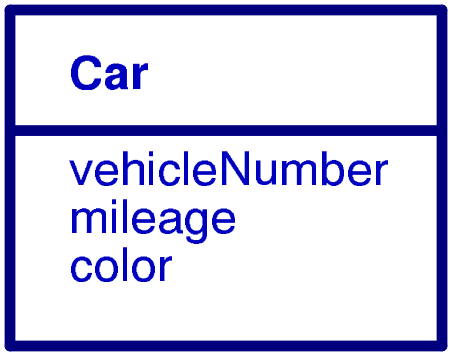


Class Diagrams



Values and Objects

- During design, you may show internal identifiers. For example, you may use internal identifiers to clarify the design of relational database tables.



OK for Analysis
OK for Design

Wrong for analysis
OK for design

Do not show object identifiers in an analysis model.

Links

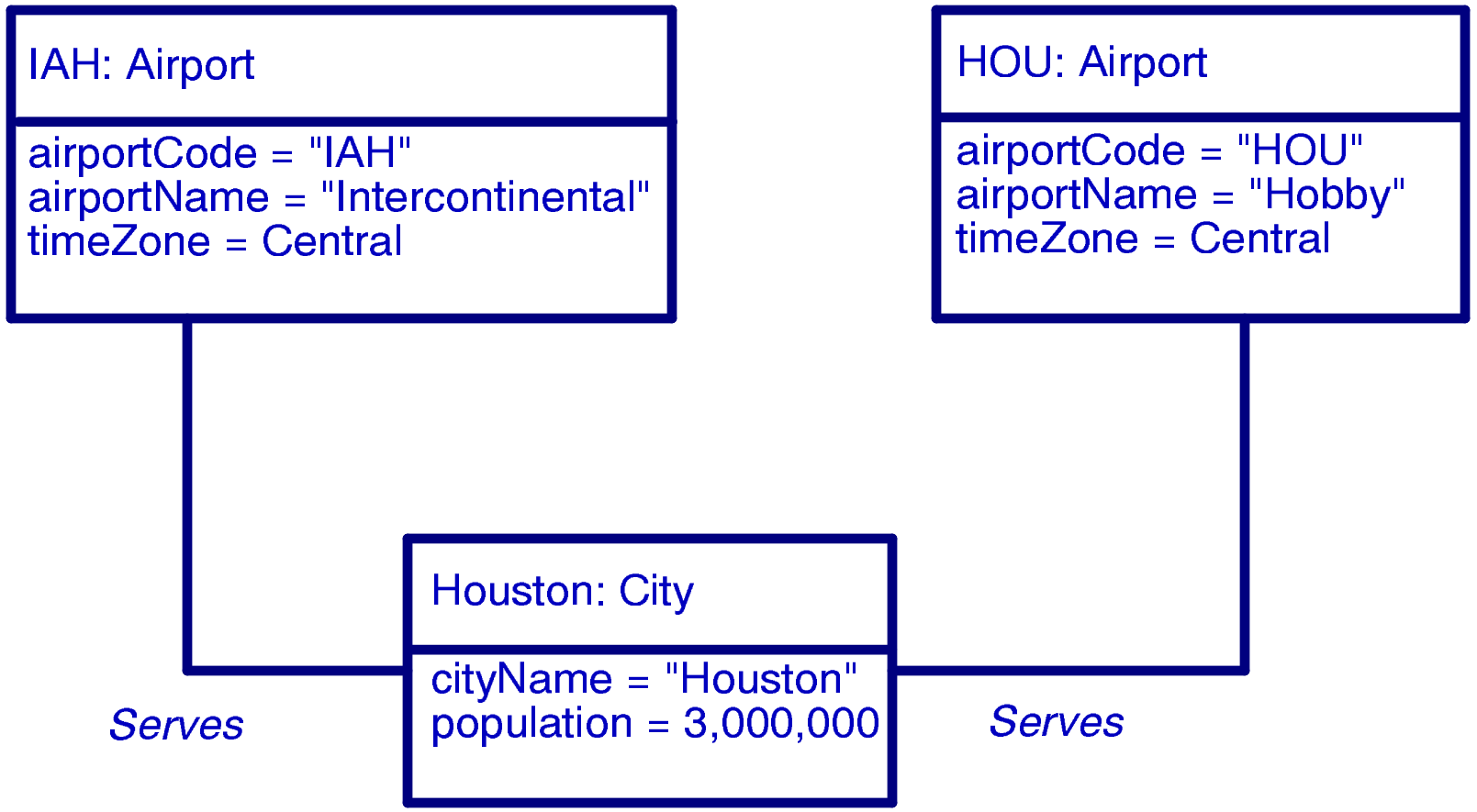
and

Associations

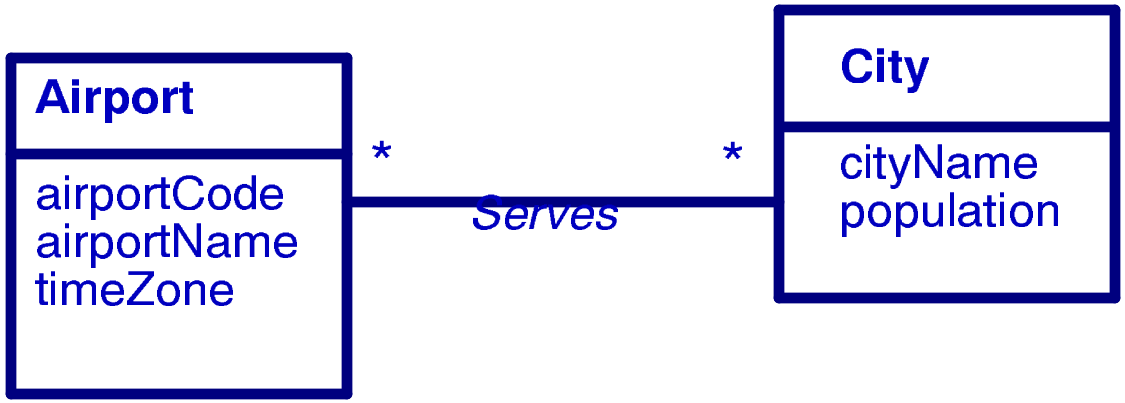
Links and Associations

- A **link** is a physical or conceptual connection between objects.
  - most relate two objects
  - some relate three or more objects
  - is not a value
- An **association** is a description of a group of links with common structure and common semantics.
- A link is an instance of an association.
- The links of an association relate objects from the same classes and have similar properties (link attributes).

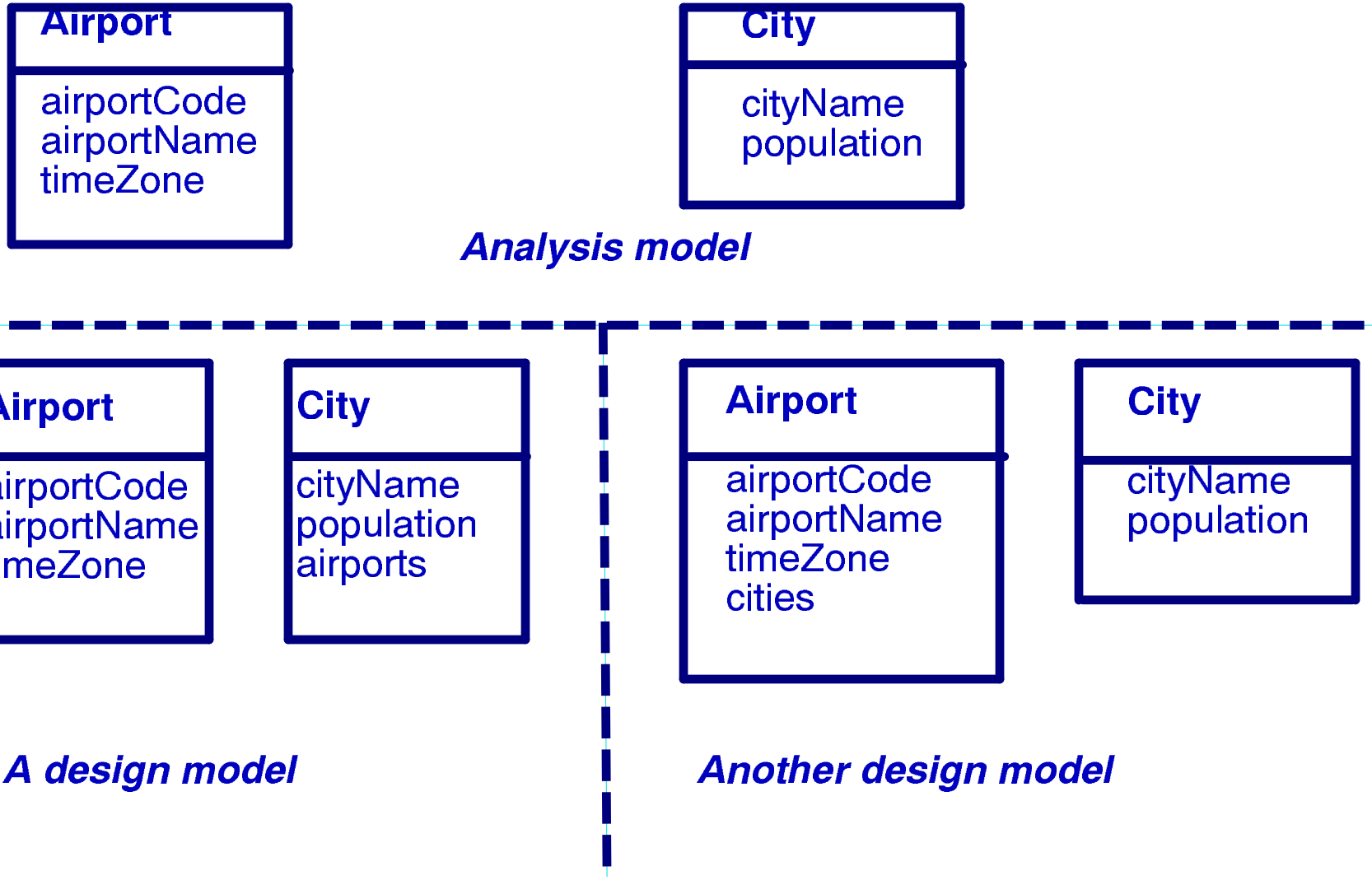
Links


IAH: Airport
airportCode = "IAH"
airportName = "Intercontinental"
timeZone = Central
HOU: Airport
airportCode = "HOU"
airportName = "Hobby"
timeZone = Central
Houston: City
cityName = "Houston"
population = 3,000,000
Serves
Serves

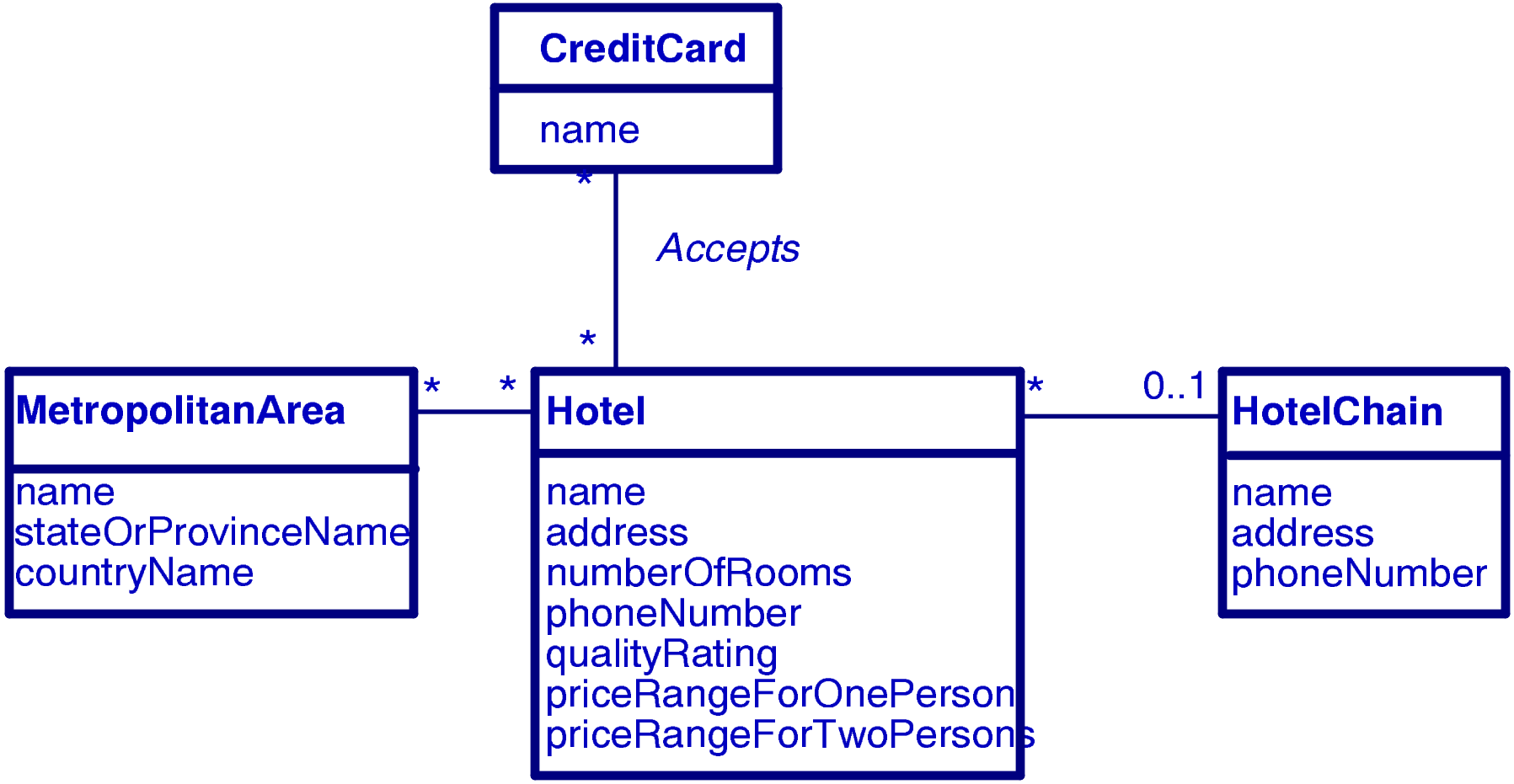
Associations



Associations
Analysis and Design Models

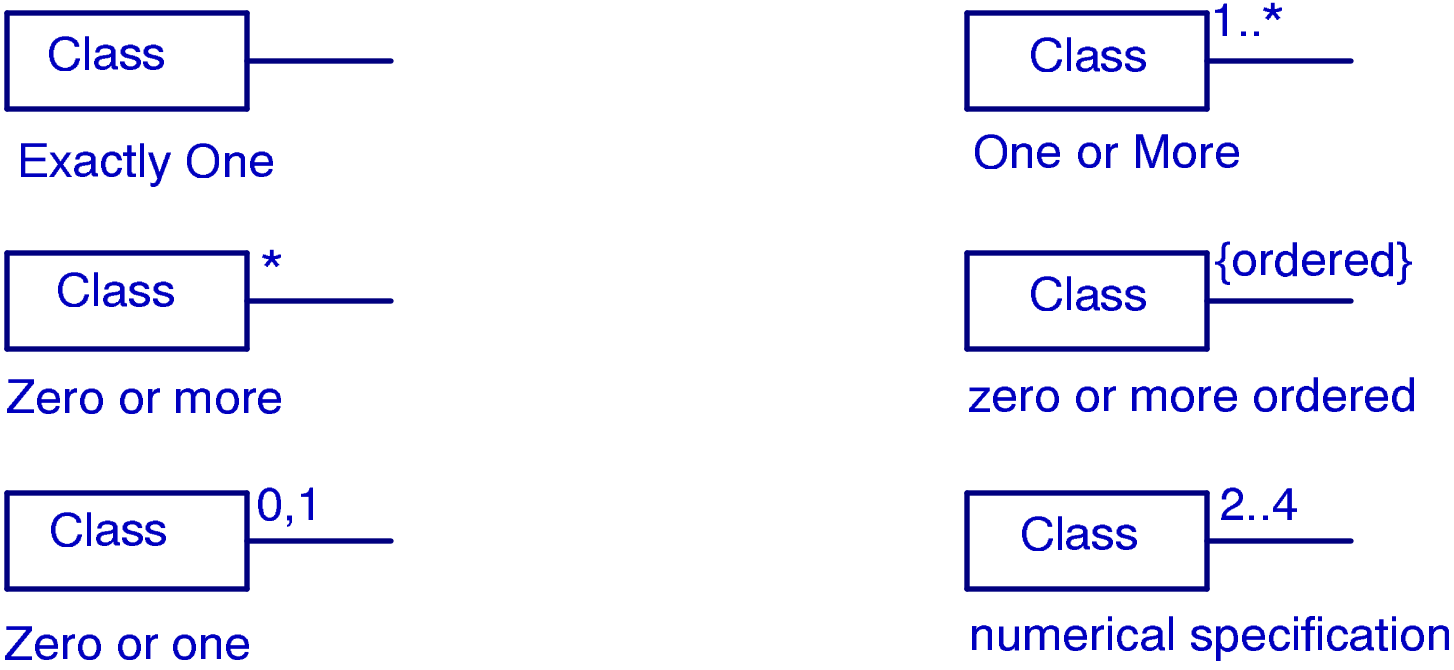


Associations in an Object Model



Associations in an object model for hotel selection.

Multiplicity



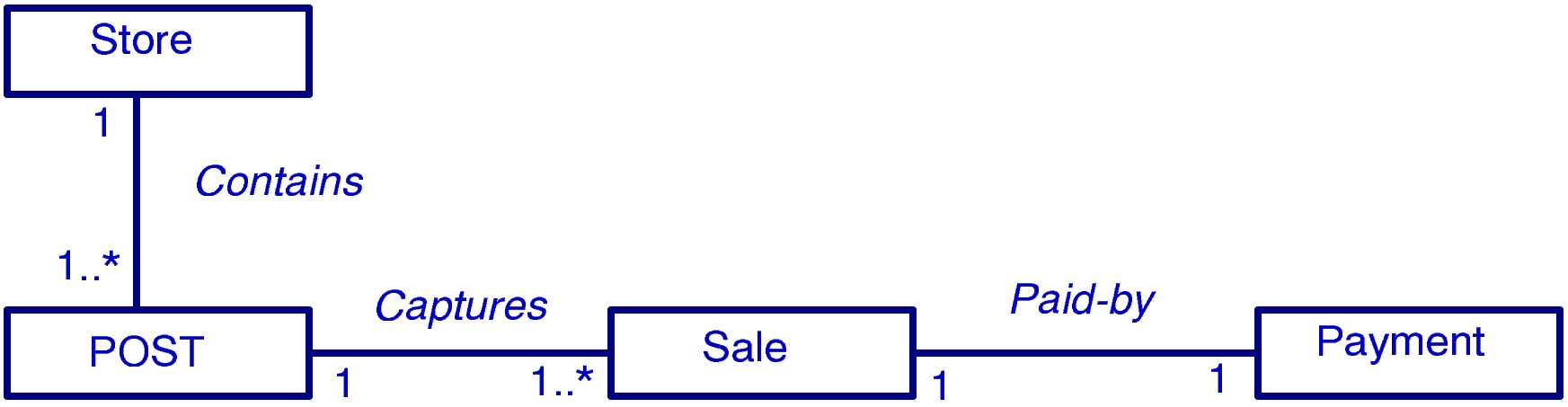
Labeling multiplicity of classes in object models.

Roles

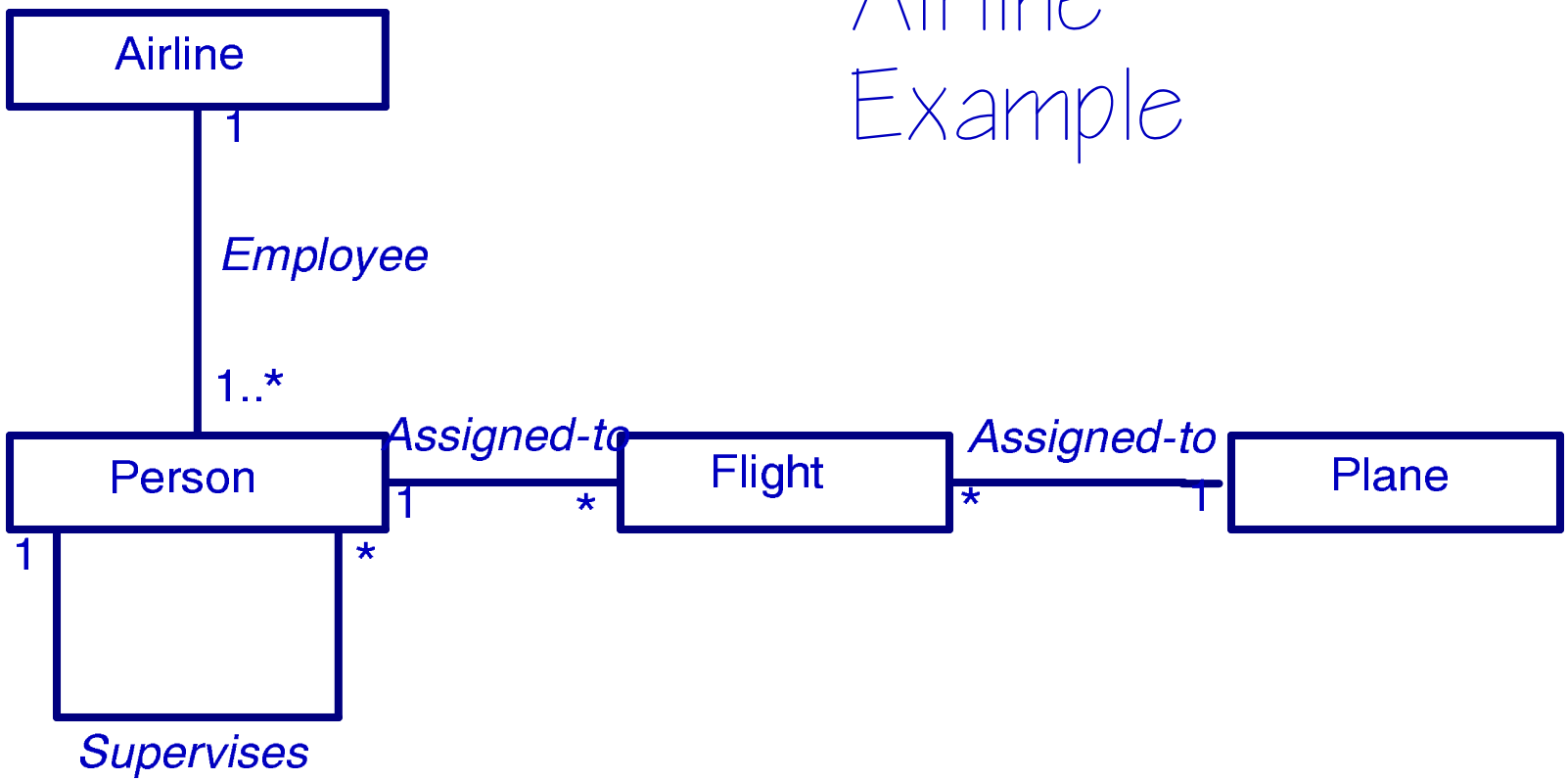
- Each end of an association is called a role.
- Roles may optionally have:
  - name
  - multiplicity expression
  - navigability

Naming Associations

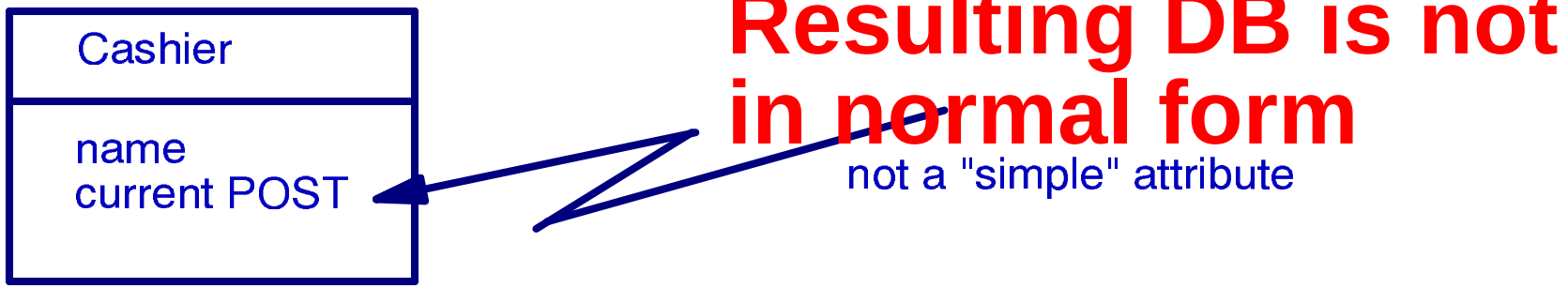
- Name an association based on a **ClassName-VerbPhrase-ClassName** format where the verb phrase creates a sequence that is readable and meaningful in the model context.
- Store Example



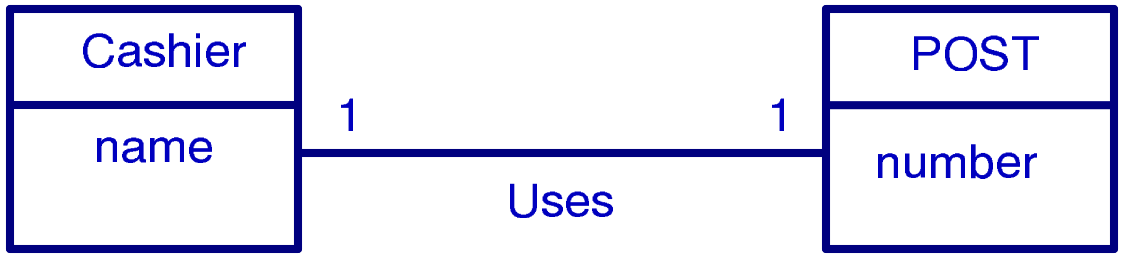
Association Names



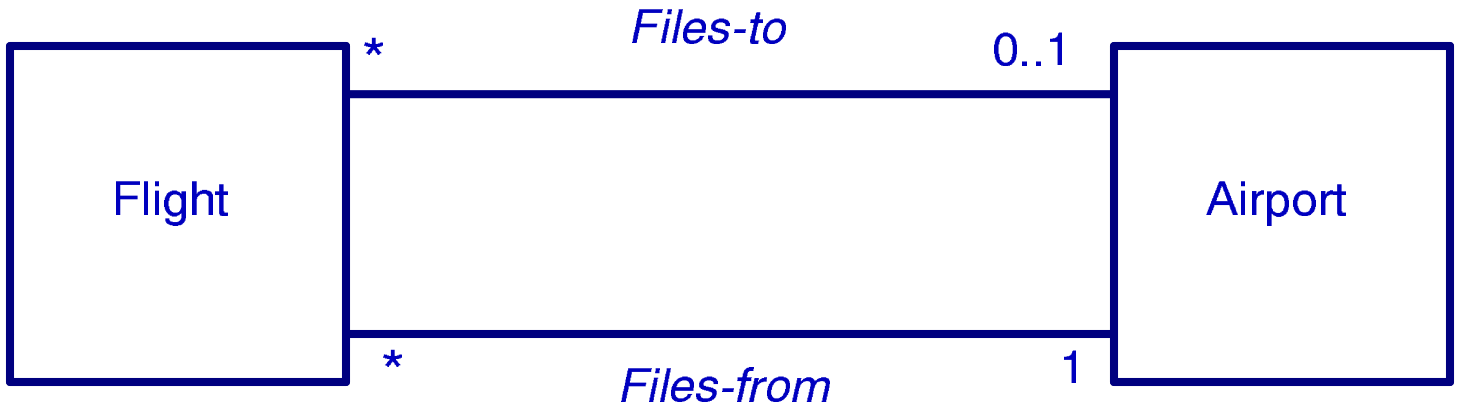
Relate with Associations not Attributes



Better



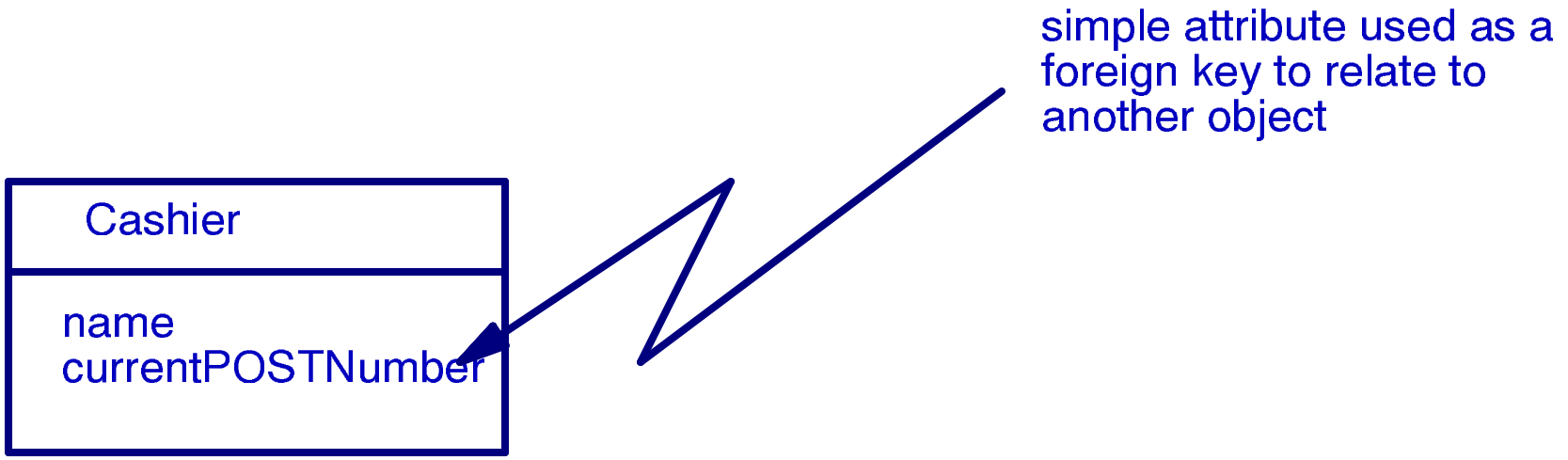
Multiple Associations Between Classes



(Not every flight is guaranteed to land at an airport!)

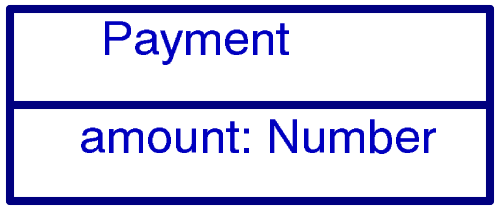
Design Creep: No Attributes as Foreign Keys

- Attributes should not be used to relate concepts in conceptual models or models created for database systems.
- **DO NOT** add foreign key attributes to associate two types -- use an association instead.

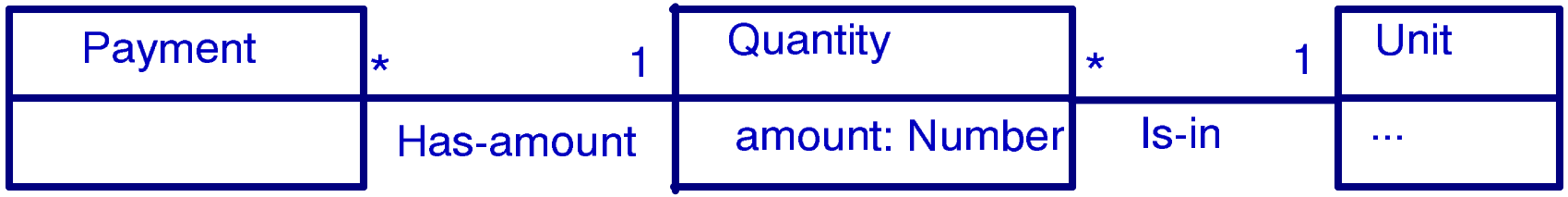


POOR DESIGN
Use a Foreign Key to relate to another object.

Modeling Quantities

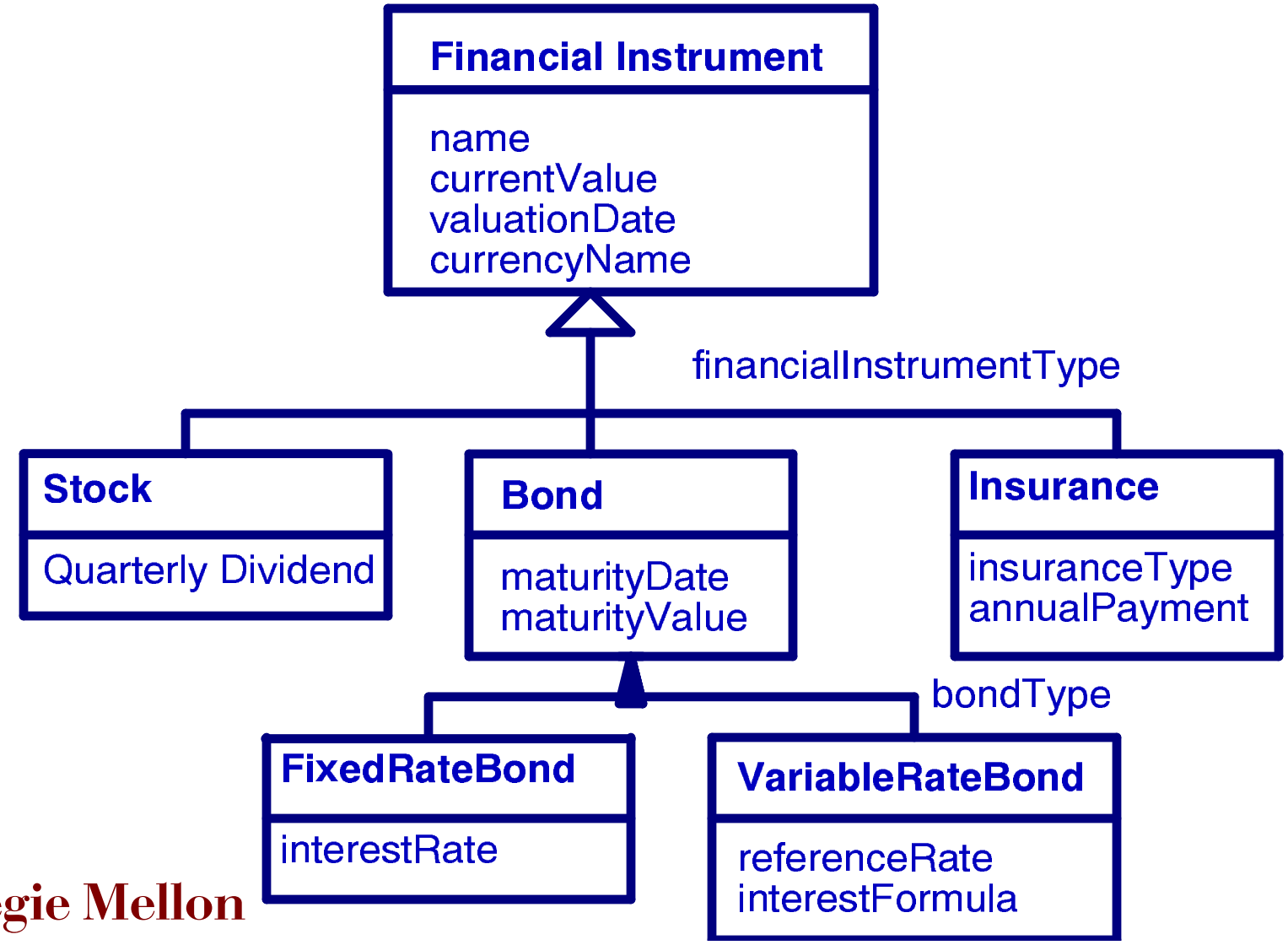


Usable, but not flexible or robust

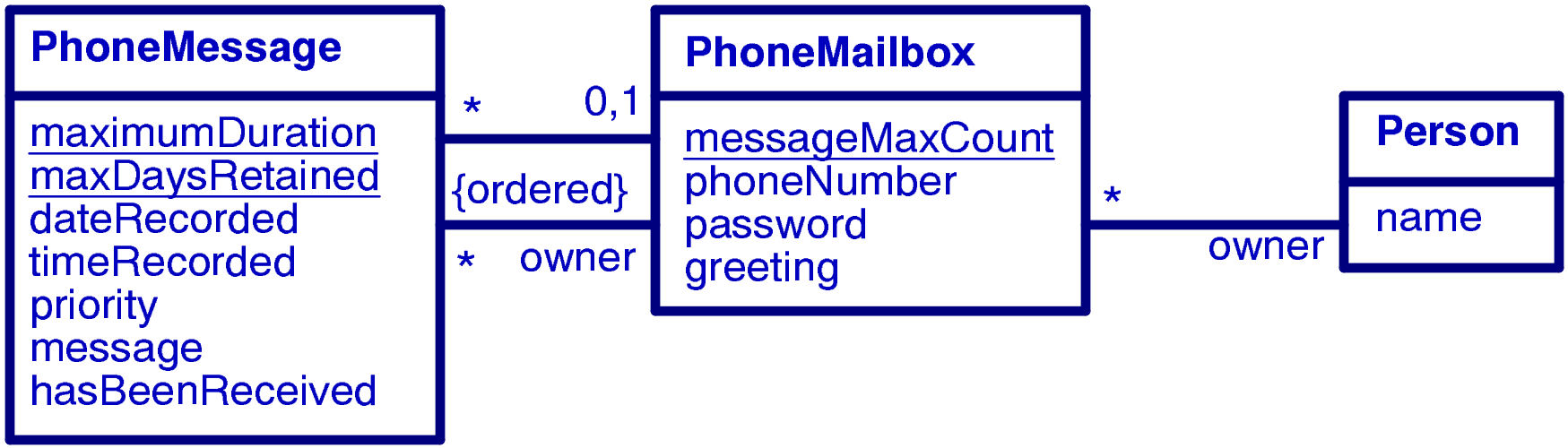


Better

Generalization



Model Groups Explicitly



POOR DESIGN

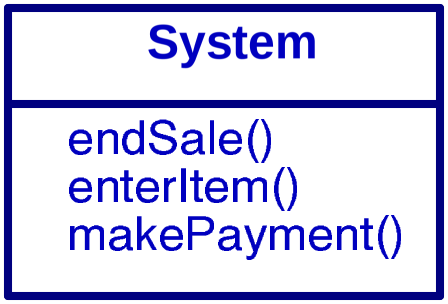
All users have same
- maximumDuration
- maxDaysRetained
- messageMaxCount

Can't change these for individual users or for periods of time

Contracts

Contract

- Document that describes what an operation commits to achieve.
- Declarative in style
- Emphasize what will happen
- System Operation Contract
  - describes changes in the sate of the overall system when an operation is executed.



enterItem Contract

Name:	enterItem(upc: number, quantity: integer)
Responsibilities:	Enter (record) sale of an item and add it to the sale . Display the item description and price.
Type:	System
Cross Refs:	System Functions: R1.1, R1.3, R1.9 Use Cases: Buy Items
Notes:	Use super fast database access.
Exceptions:	If the UPC is not valid, indicate that it was an error.
Output:	
Pre-Conditions:	UPC is known to the system.
Post-conditions:	Next slide.

Post Conditions

- Useful categories (again):
  - Instance Creation and Deletion
  - Attribute Modification
  - Links formed and broken
- Post Conditions described in terms of the conceptual model
- Describe only what is done, not how!!

enterItem Post-Conditions

- If a new sale, a Sale was created (instance creation)
- If a new sale, the new Sale was linked to the POST (link formed).
- A SalesLineItem was created (instance creation).
- The SalesLineItem was linked to the Sale (link formed).
- SalesLineItem.quantity was set to quantity (attribute modification).
- The SalesLineItem was linked to a ProductSpecification, based on UPC match (link formed).

Advice for Creating Contracts

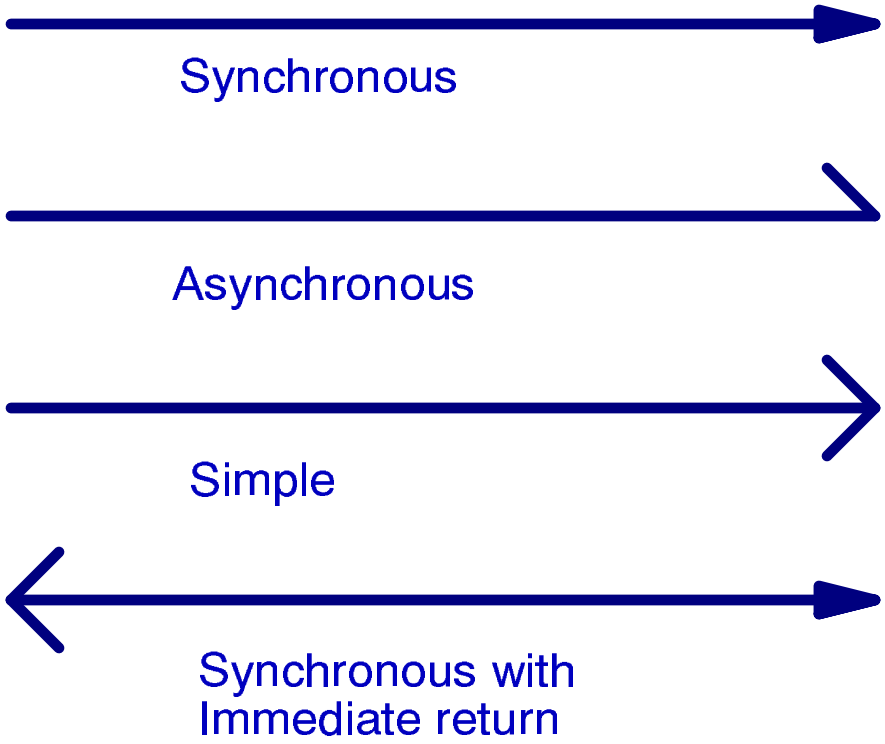
- Identify system operations from sequence diagrams.
- Construct a Contract for each system operation
- Start by writing the Responsibilities Section (informal description of the operation).
- Complete Post-Condition Section
  - describe state changes of objects declaratively
- To describe Post Conditions
  - instance creation and deletion
  - Attribute modification
  - Links formed and broken
- Add preconditions

Contract for endSale()

Name:	endSale()
Responsibilities:	Record that it is the end of entry of sale items and display sale total.
Type:	System
Cross Refs:	System Functions: R1.1, R1.3, R1.9
Notes:	
Exceptions:	If a sale is not underway, indicate that it was an error.
Output:	
Pre-conditions:	UPC is known to the system.
Post-conditions:	*Sale.isComplete* was set to *true* (attribute modification)

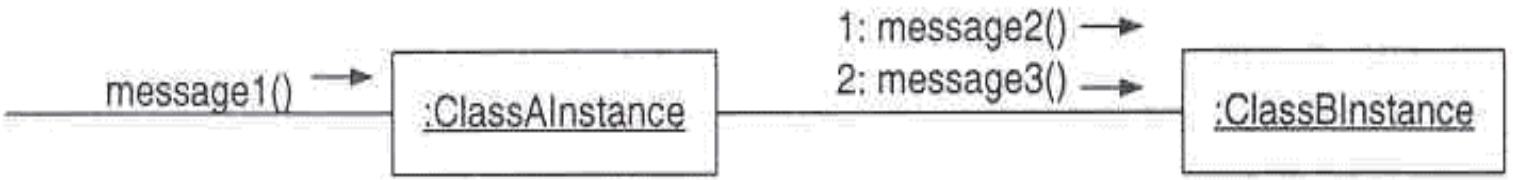
Interaction Diagrams

Message types notation

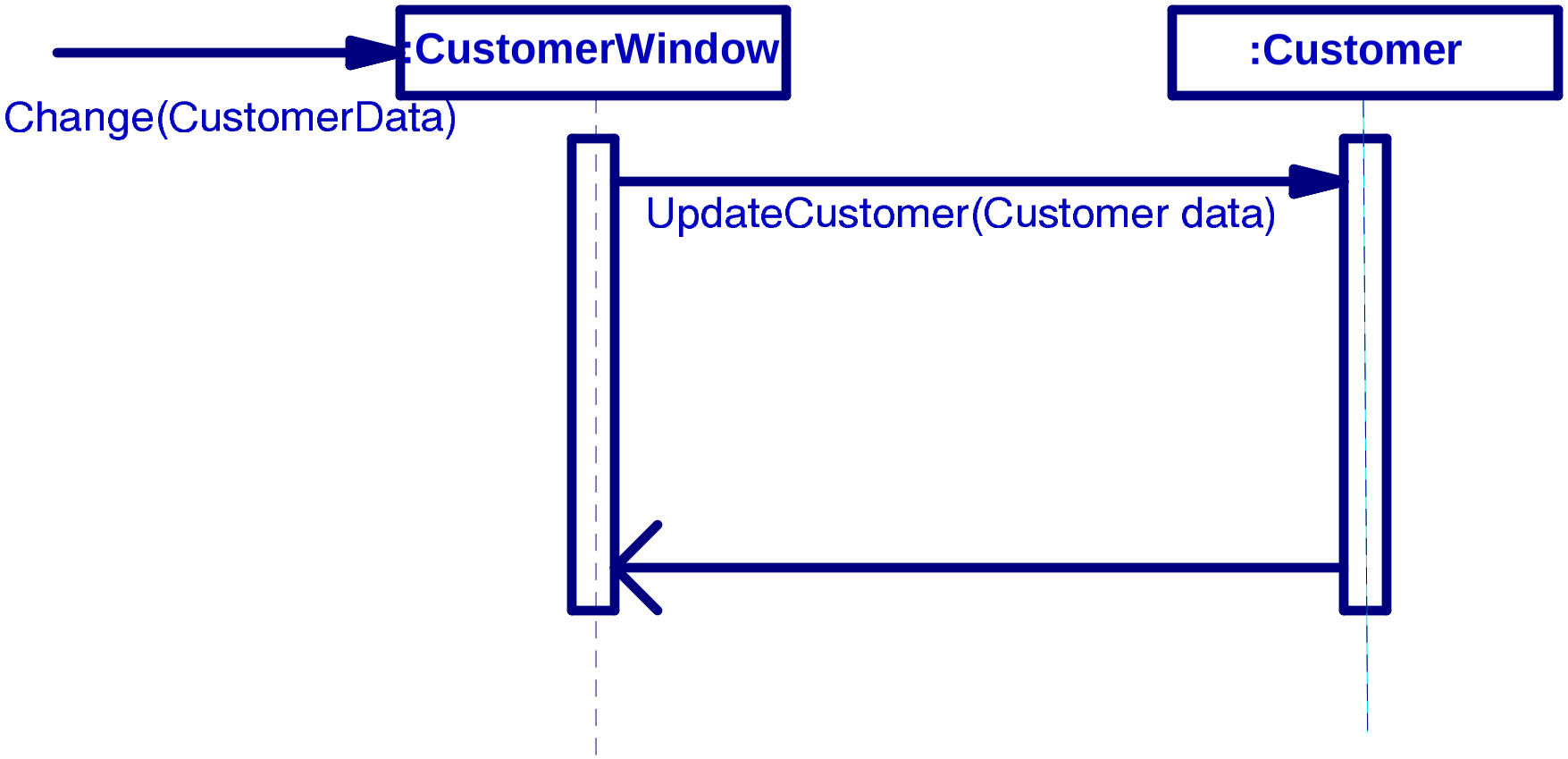


Simple Collaboration Diagram

- Shows messages sent to and by objects

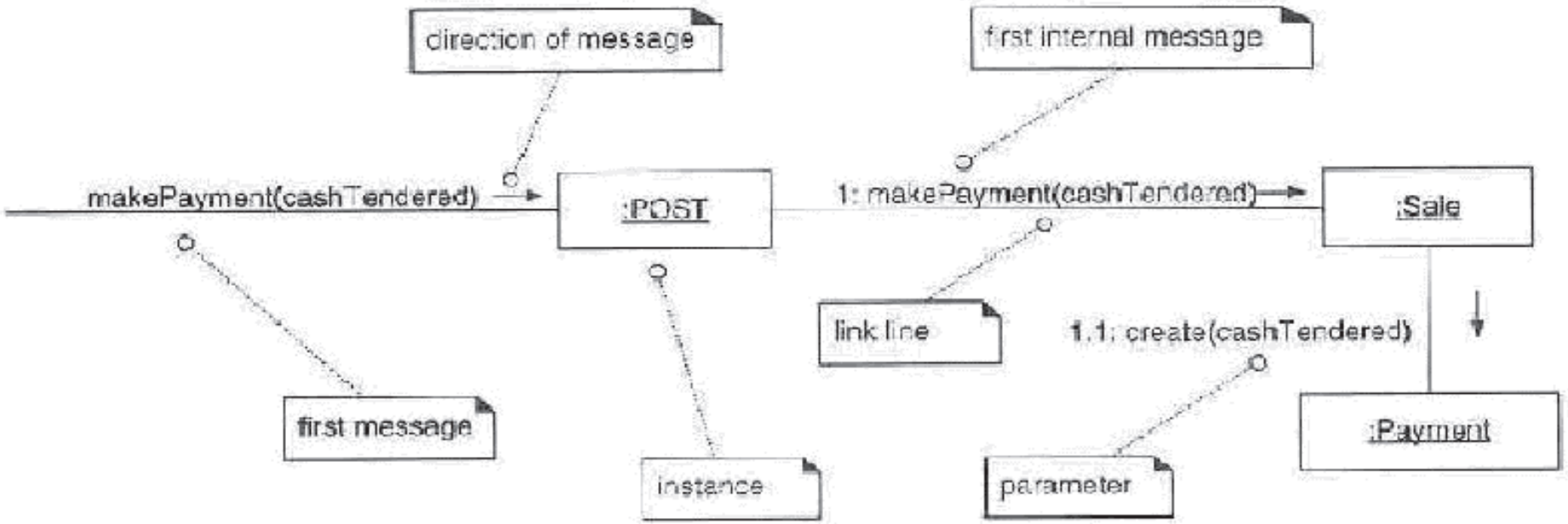


Simple Sequence Diagram



Example Collaboration Diagram

- Message makePayment is sent to instance of **POST**
  - corresponds to the **makePayment** system operation message
- The POST object sends the makePayment message to a Sale instance.
- The Sale object creates an instance of a Payment.



Guidelines for Making Collaboration Diagrams

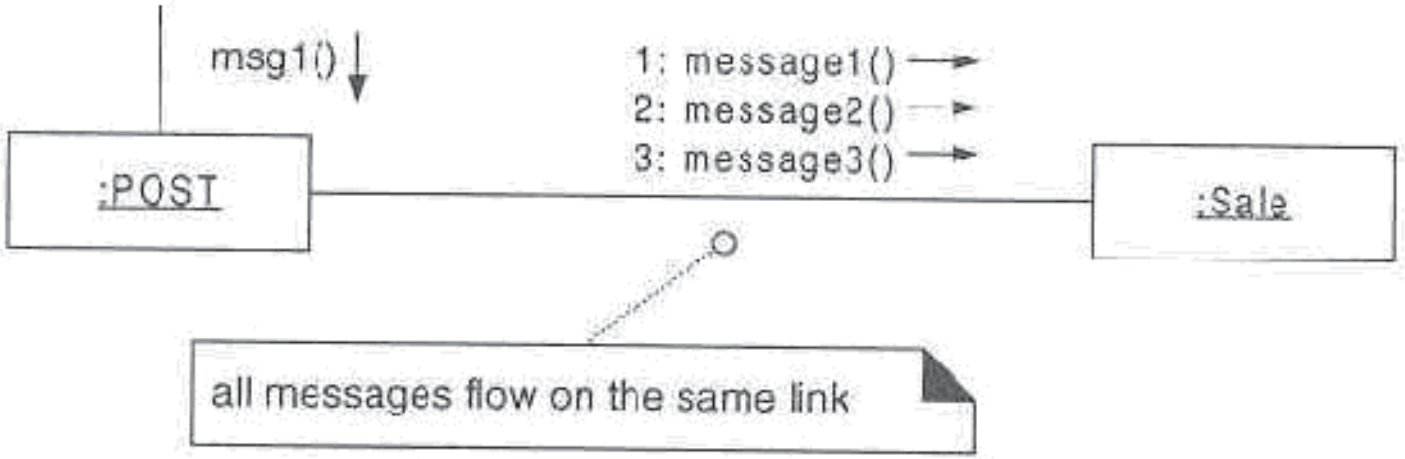
- Create separate diagram for each system operation being developed in the current cycle
  - For each system operation message, make a diagram with it as the starting message
- If a diagram gets complex, split it into smaller diagrams
  - A diagram should fit on a sheet of 8.5 by 11 paper
- Using
  - operation contract responsibilities
  - contract post-conditions
  - use case descriptions
  - use case diagrams
- design a system of interacting objects to complete the tasks
- GRASP from next lecture helps a lot.

Showing Links

- Link is a connection path between two instances
  - indicates some form of
    - navigation
    - visibility
  - is possible
- A link is an instance of an association
- Link from a POST to a Sale indicates messages can be sent from POST instance to Sale instance.

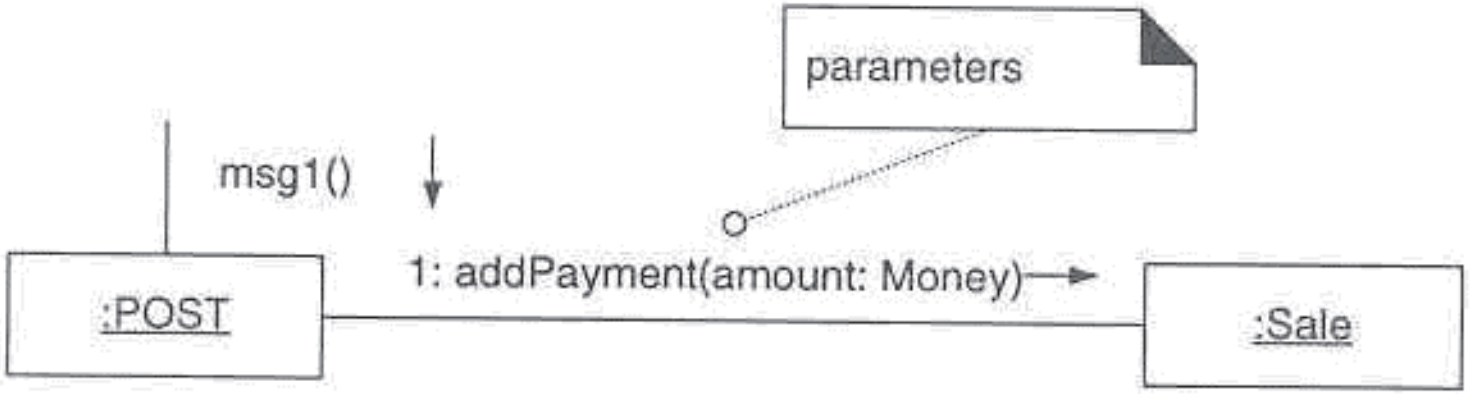
Showing Message

- Messages are a labeled arrow on a link line
- Any number of messages my flow along this link
- Sequence number shows the order of messages in the current thread of control



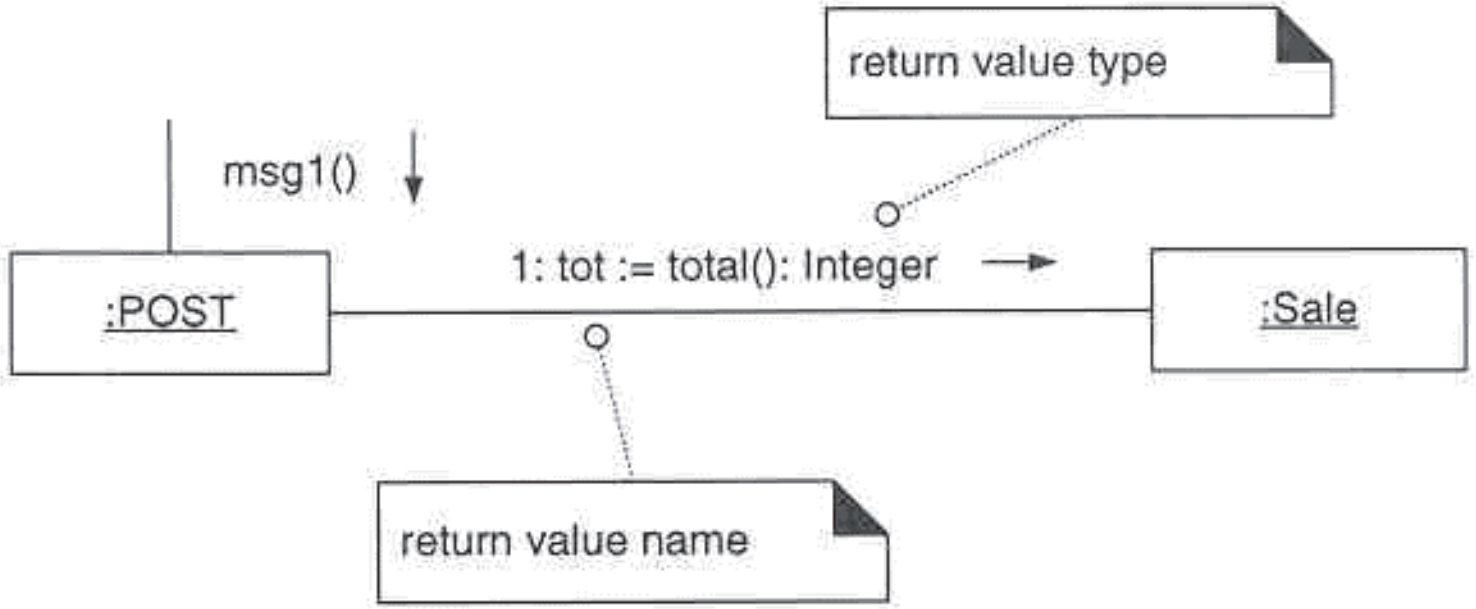
Showing Parameters

- Shown in parentheses following the message name
- Type of parameter may optional be shown



Illustrating a Return Value

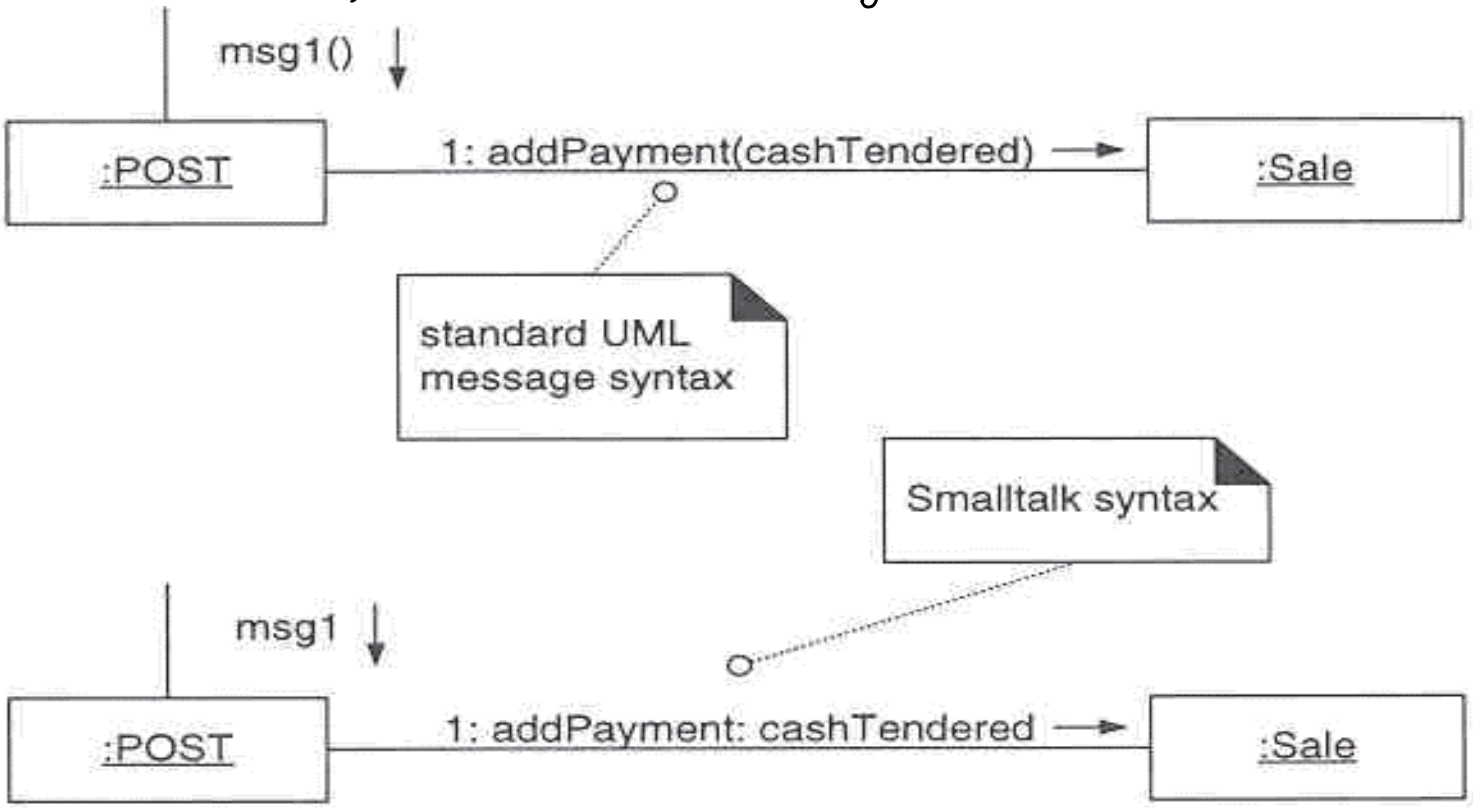
- Return value shown by preceding message with a return value variable name and an assignment operator.



More on Messages

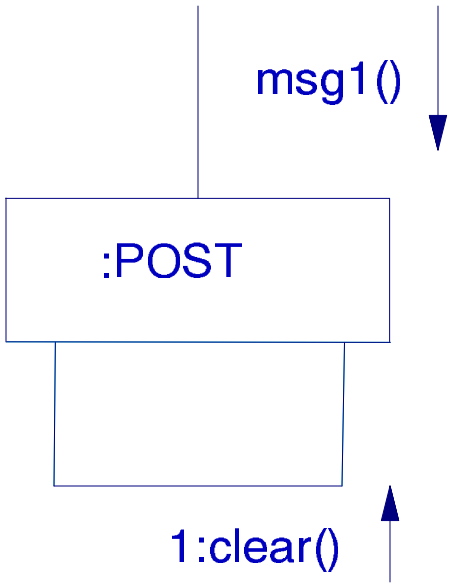
Message Syntax

- Standard syntax for messages
  - return := message(parameter : parameterType) : returnType
- Can also use Java, C++ or Smalltalk syntax



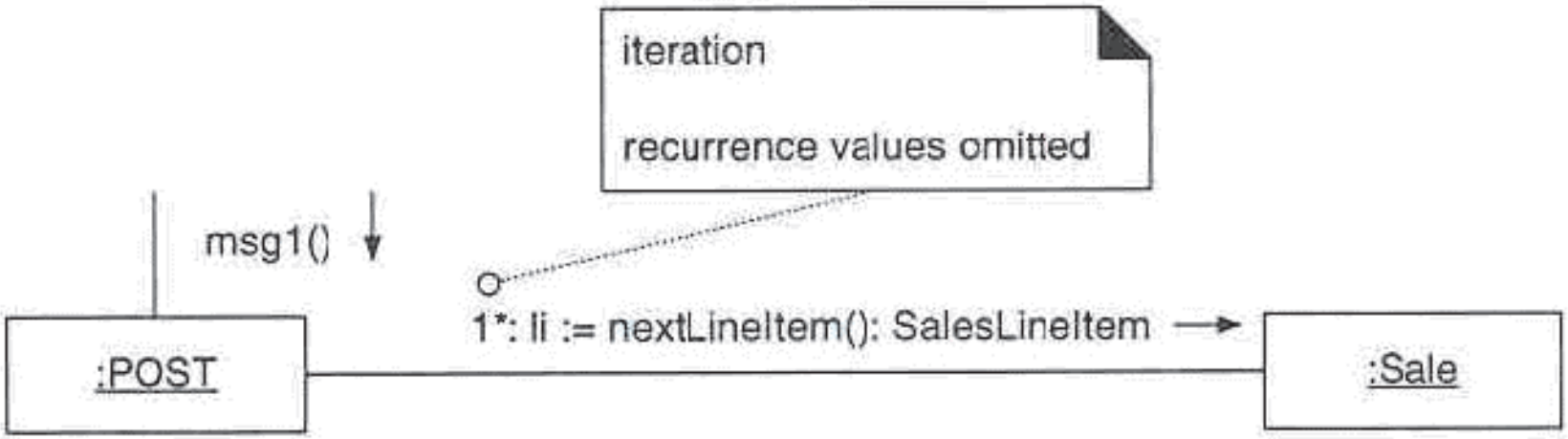
Messages to "self"

- Message can be sent from an object to itself
- Shown as link to itself, with messages flowing along the link



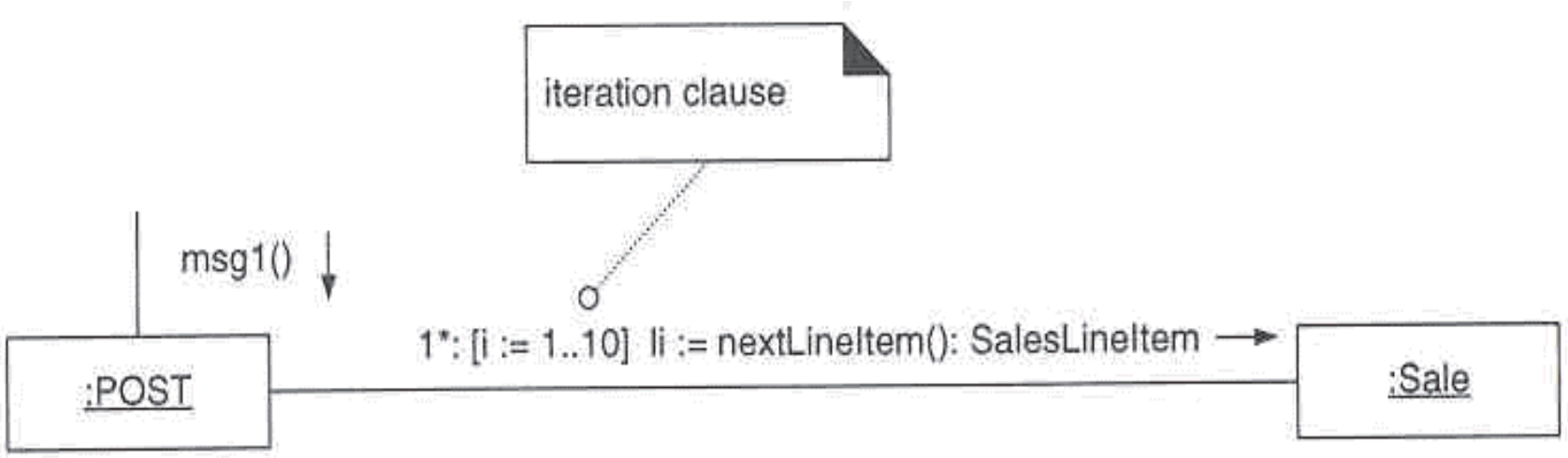
Iteration

- Iteration indicated by star (*) following sequence number
- Expresses that message is being sent repeatedly.



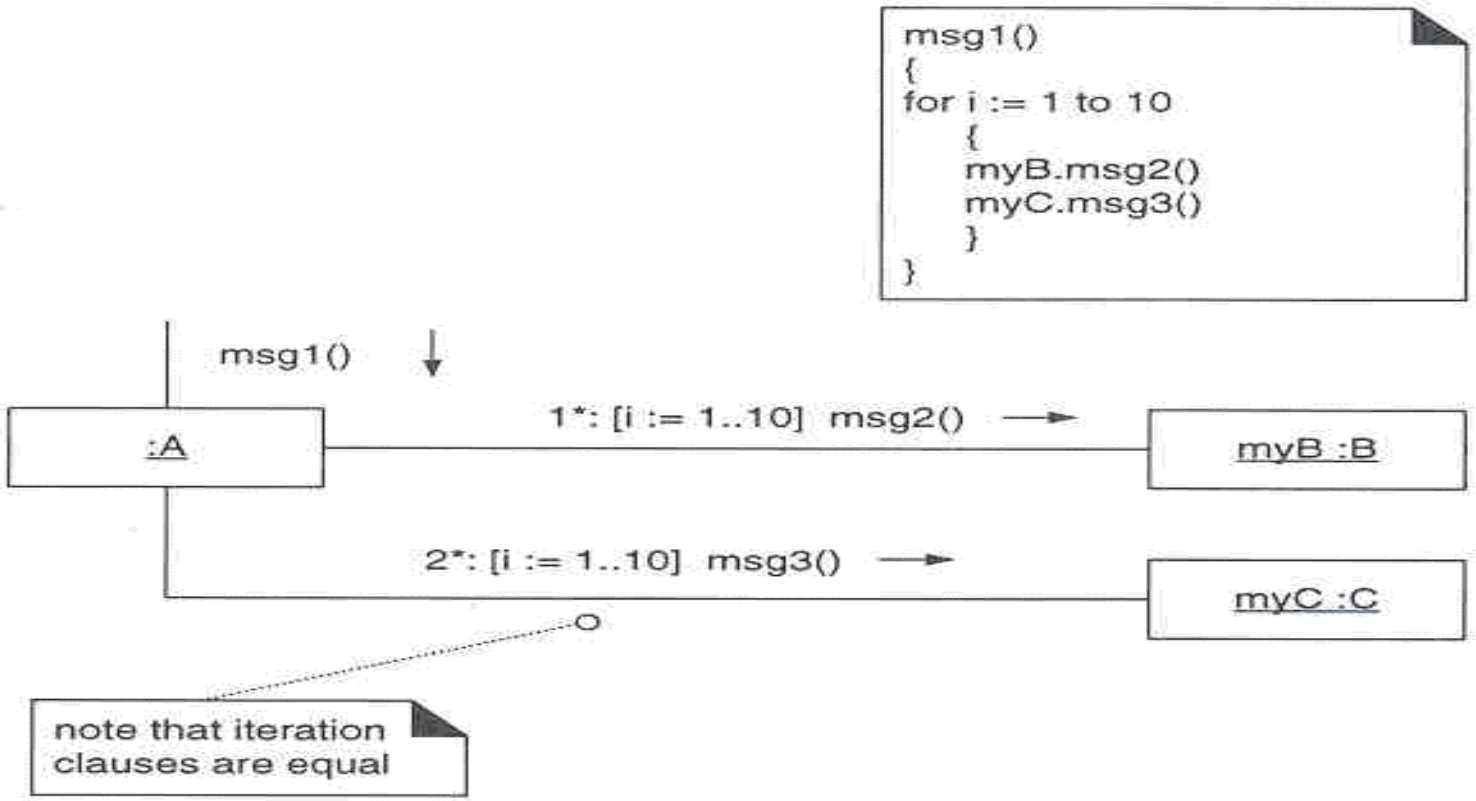
Iteration Clause

- Can add iteration clause to indicate recurrence value

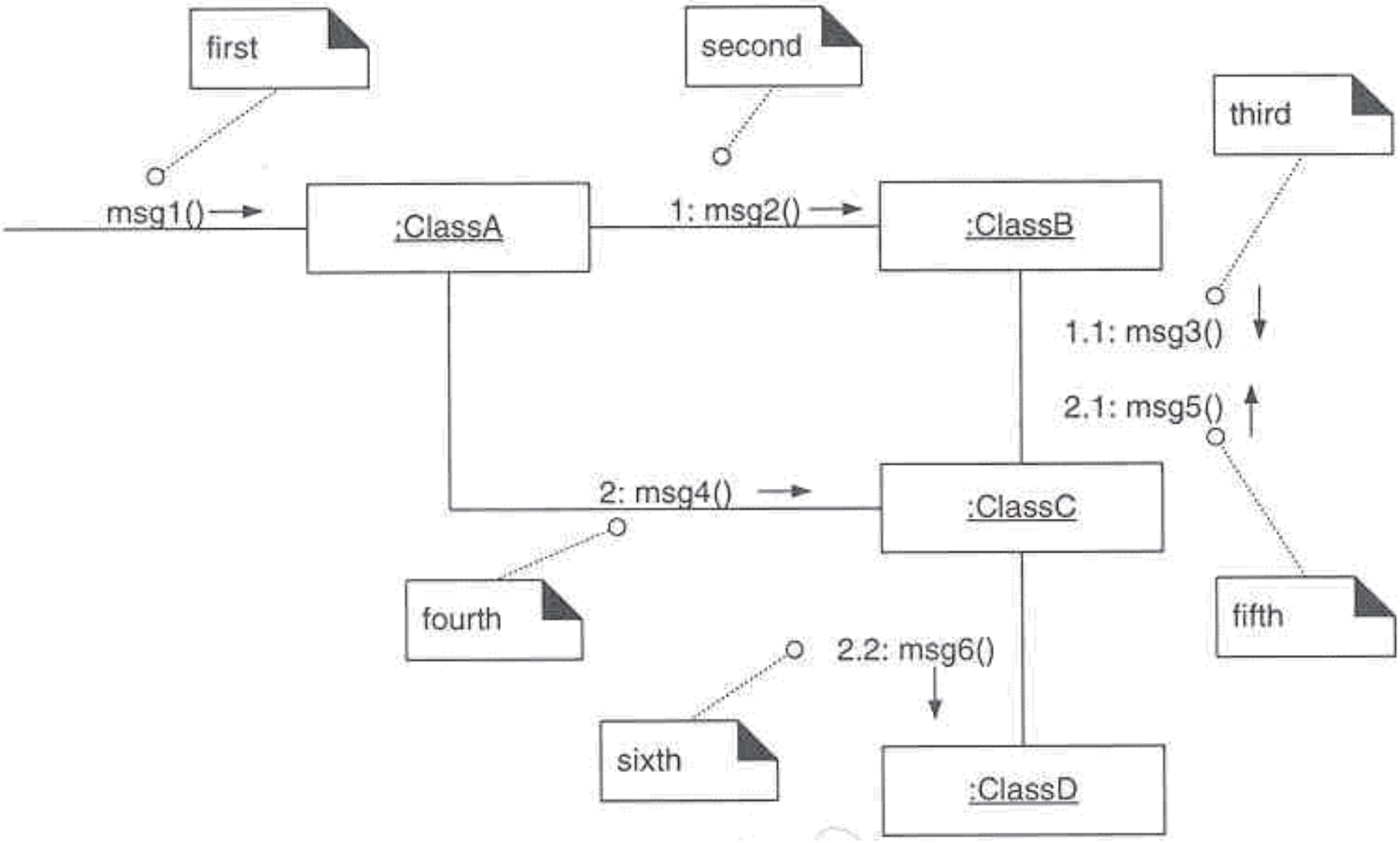


More than One Message in Iteration Clause

- To express more than one message happening within the same iteration clause, repeat the iteration clause on each message

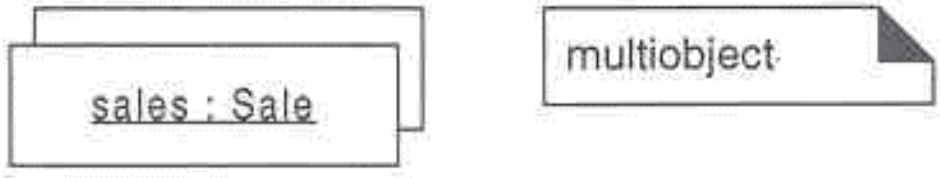


Message Sequence Numbering II



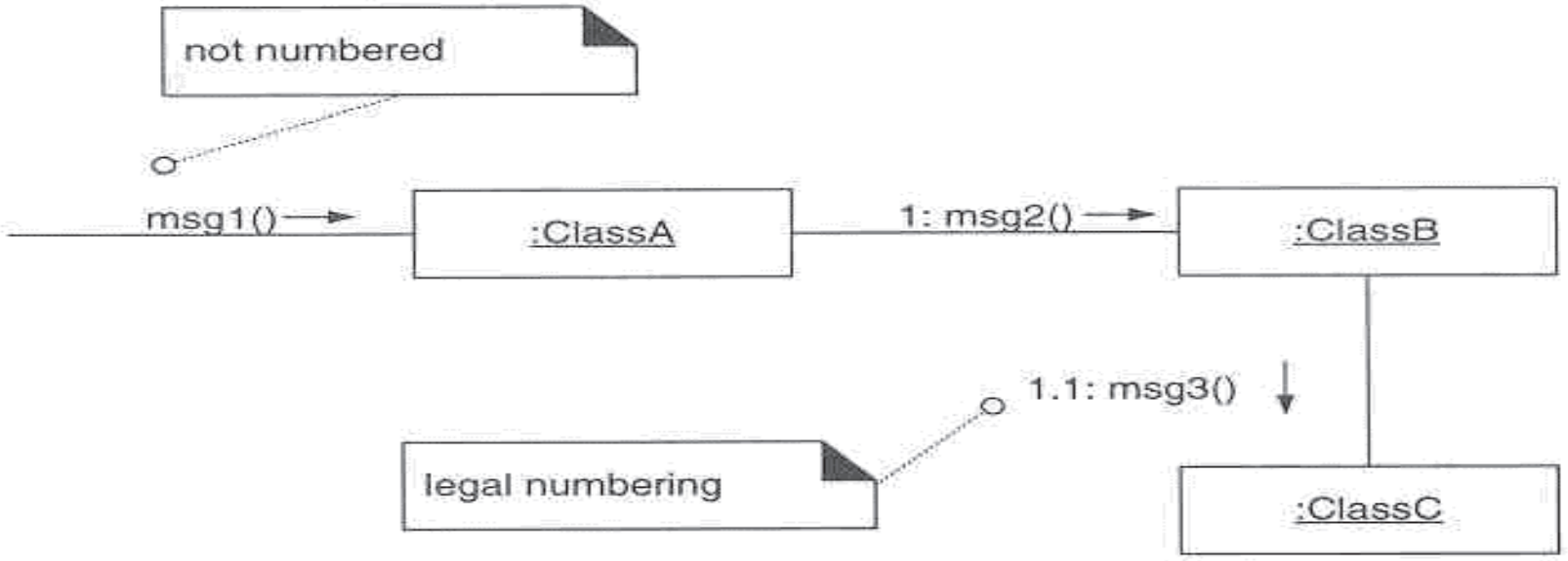
Collections

- A **multiobject** or set of instances shown with stack icon
- Usually implemented as a group of instances stored in a container object, e.g., Java Vector or Smalltalk OrderedCollection



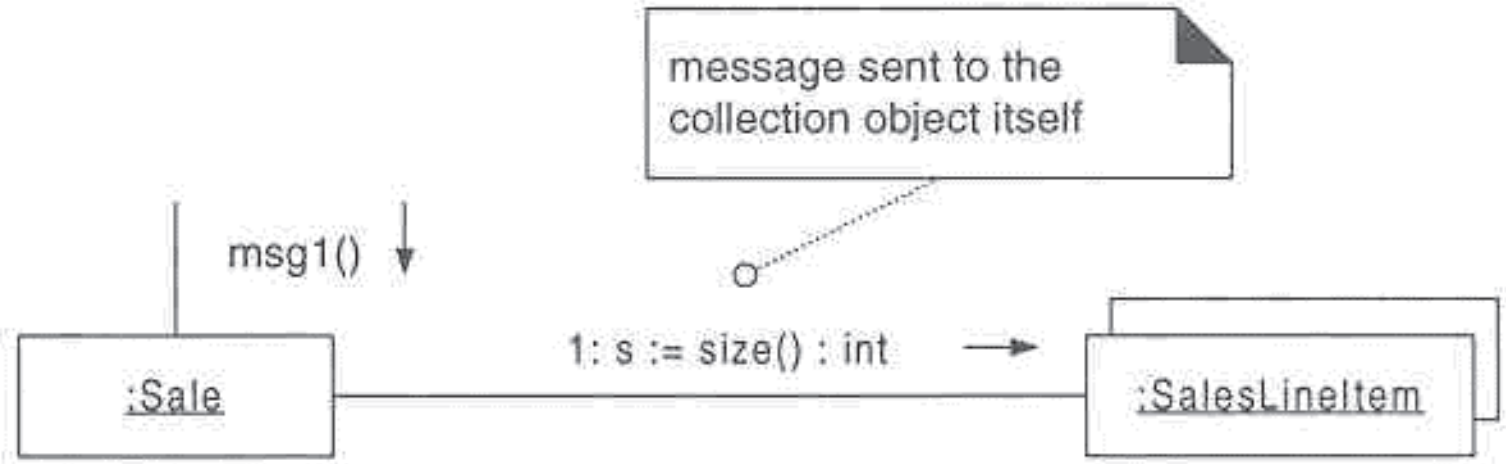
Message Number Sequencing

- Order of messages is shown with **sequence numbers**
  - First message is unnumbered
    - msg1() is unnumbered
  - Order and nesting of subsequence messages is shown
    - nested messages have a number appended
    - nesting denoted by prepending the incoming message number to the outgoing message number



Messages to Multiobjects

- Message sent to multiobject icon is sent to the collection object that contains the objects
- Example shows size message is being sent to a Java.util.Vector instance to query the number of elements in the vector.



Messages to Multiobjects -- 2

- Messages to a multiobject and an element