UNIVERSITY OF CAPE TOWN
Department of Computer Science

Advanced Software Design: Review of Object-Oriented Modelling, UML and Analysis

JonJon Clark
clrjon005@myuct.ac.za

slides and materials taken from prof edwin blake & Melissa Densmore

23 July 2018

Object oriented modelling, use case writing and UML

- **Background**, catch up for yourself if anything is unfamiliar!
- Blue triangle here ≡ none of this stuff is *directly* examinable
  - of course an understanding is required to do the practical work and understand the new material in the course
  - and so in an exam it may be indirectly examined.

OBJECT ORIENTATION

Introduction to Object Oriented Development

Object oriented ≡ objects + classification + inheritance + communication

Inception Phase: Analysis and Use Cases

UML

Dynamic Behaviour Views

Structural Views

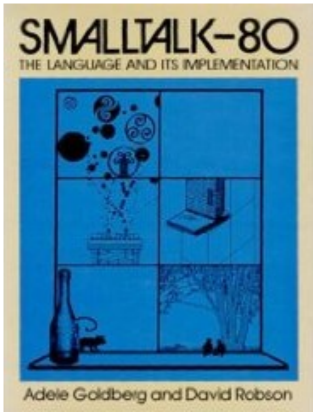
Object-oriented Programming ≡ Managing Complexity

- Complexity:
- Modelling
  - Data abstraction (objects)
  - Structure inheritance (sub-types)
- Procedural
  - Messages to invoke methods
  - Behavioural inheritance (sub-types)
- Coding
  - Meta classes (constructors/destructors)
  - Code re-use (code inheritance)

01000111 01110010 01100101
WHAT ON EARTH ARE YOU DOING?!
SPEAKING IN BINARY. I FIGURE NOW THAT THE DIGITAL AGE HAS TAKEN OVER, IT MAKES SENSE TO ADOPT WHAT IS TRULY NOW THE DOMINANT LANGUAGE.
01100101 01110100 01101001 01101110...
JASON, I CAN'T UNDERSTAND A WORD YOU'RE SAYING!
ACTUALLY, I, UM, HAVEN'T FINISHED THE FIRST WORD YET.
LOOK, COMPUTER BOY, UNLESS YOU WANT ME TO BOOT YOU...

History of Object-Oriented Methods

- Originated in a simulation language: Simula
- model physical objects
- collect common features in classes
- Object-Oriented Methods (acid test $\Leftarrow$ inheritance)
- First real O-O Language: Smalltalk
  - www.squeak.org/


SMALLTALK-80
THE LANGUAGE AND ITS IMPLEMENTATION
Adele Goldberg and David Robson

Fundamental Ideas

- Regard the parts of a model, interaction, or simulation as independent active objects (actors):
  - an internal state
  - communicating via messages
- Abstraction: fundamental principle of modelling
  - Model at different levels: lower levels have more detail
  - Look only at relevant information
  - Hiding confusing detail
- Principles of data abstraction:
  - different internal data representations & algorithms co-exist
  - multiple external views on the same object

Object Oriented Paradigm

Data abstraction (named interfaces & hidden local state)

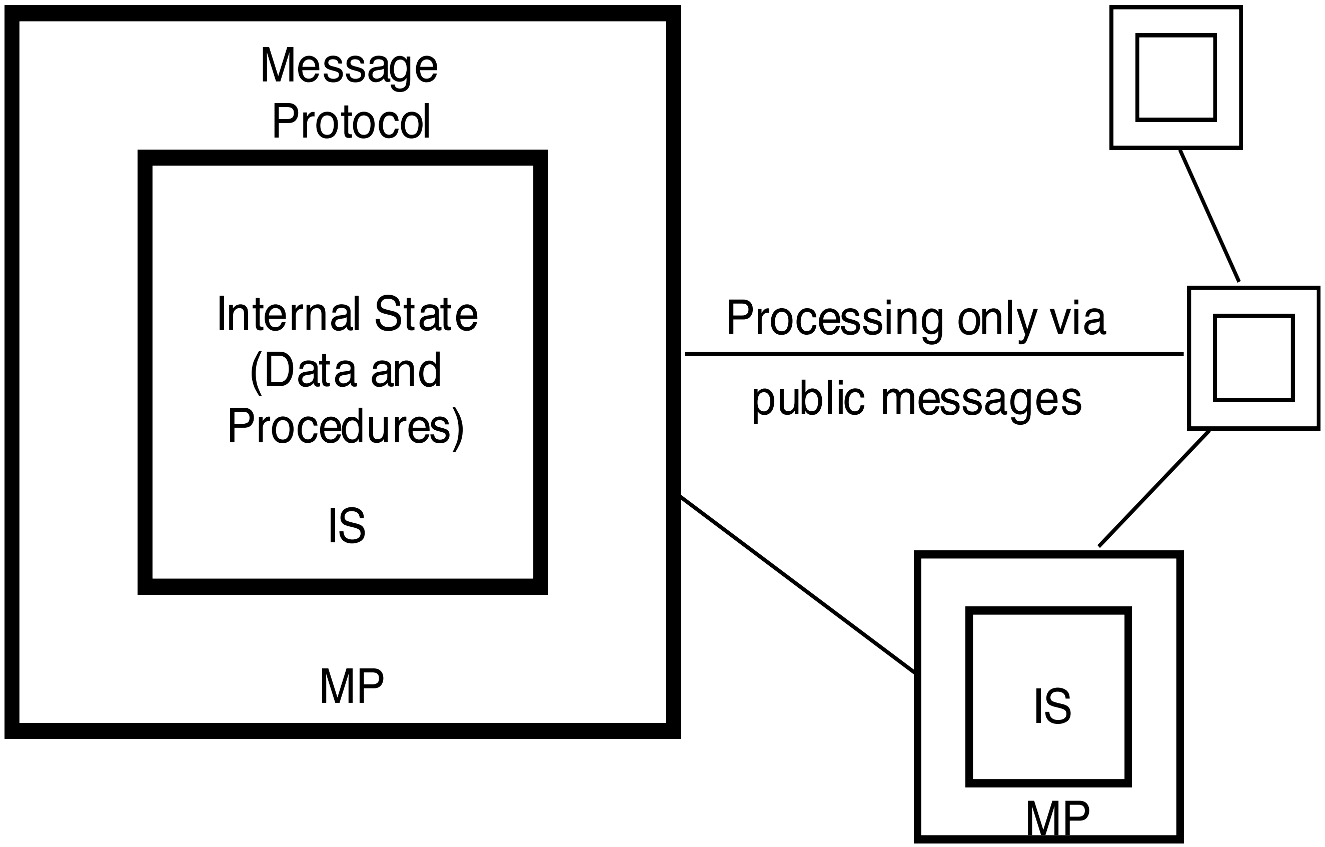
+ *object types* (or classes)

+ *type inheritance* (attributes inherited from superclasses).

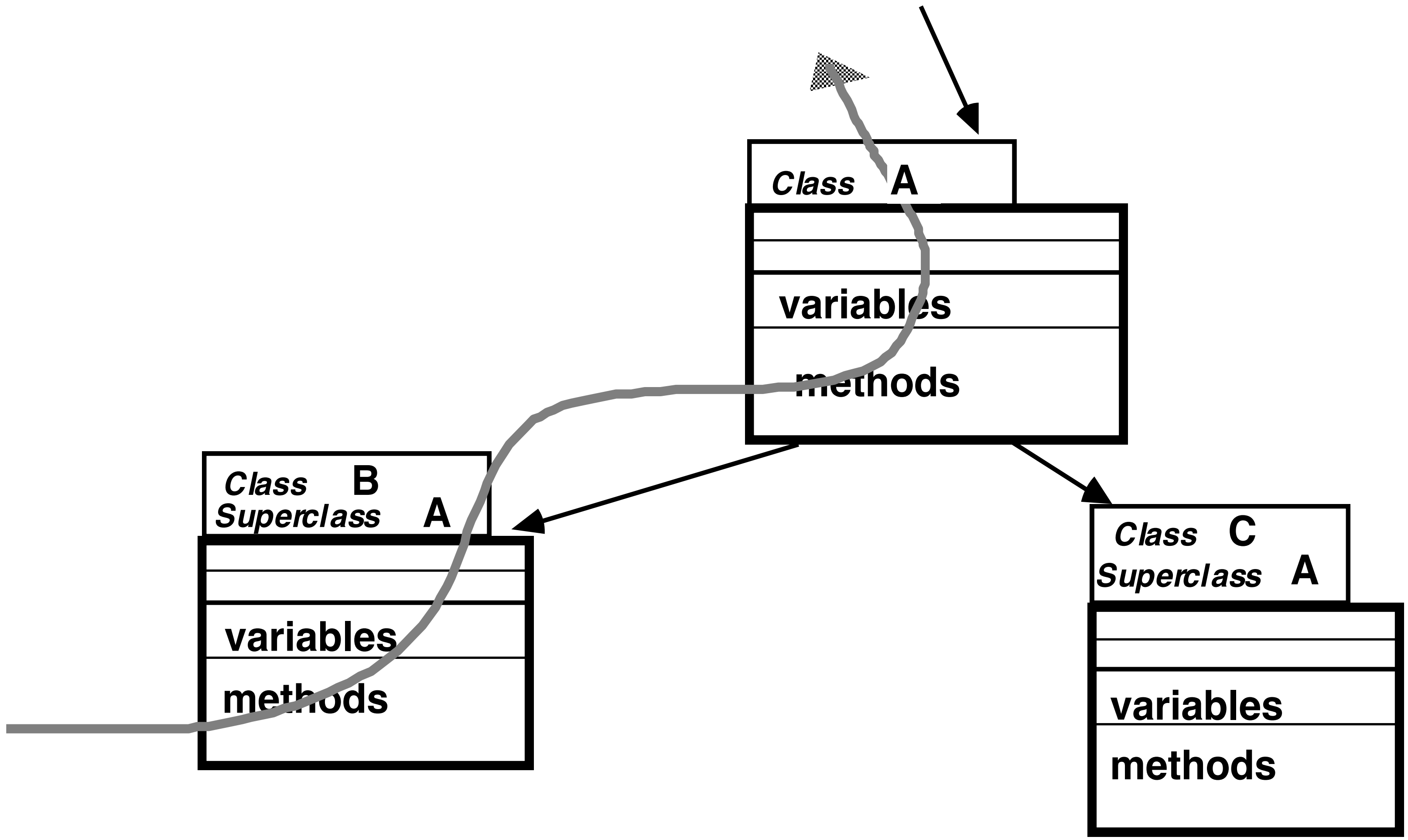
Computation by objects sending & replying to messages

Object oriented ≡ objects + classification + inheritance + communication

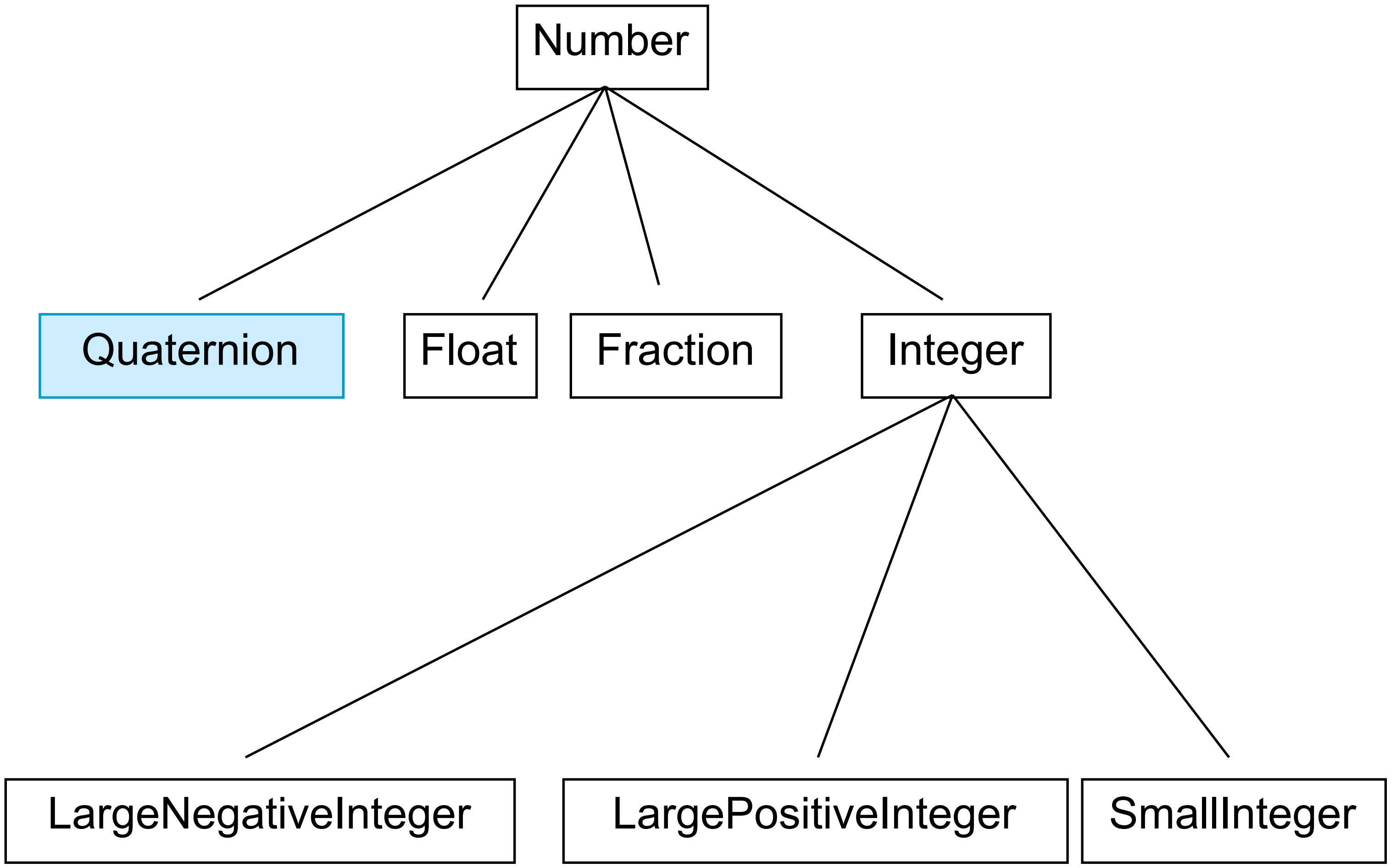
Abstract Data Types & Encapsulation



Inheritance

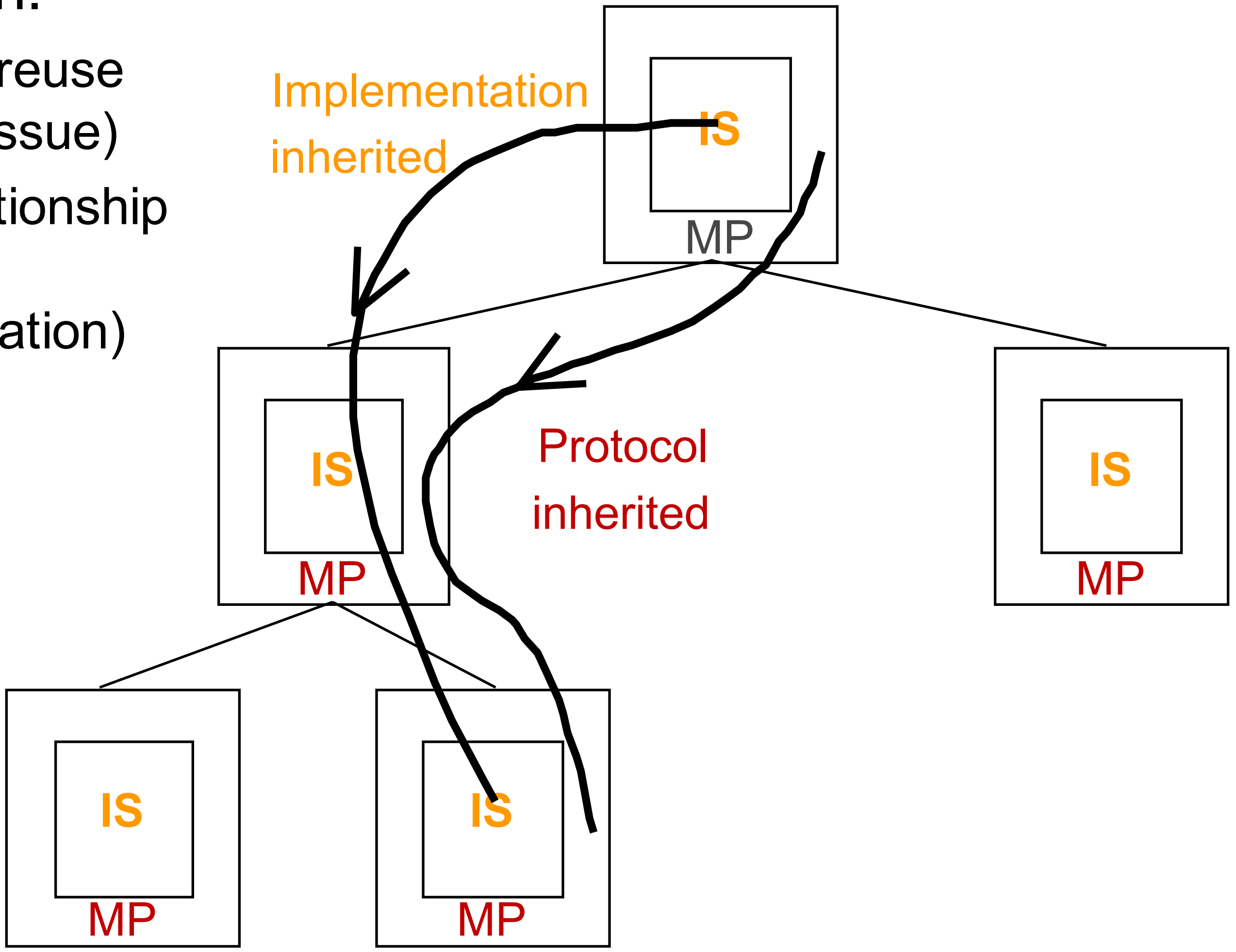


Adding a New Type of Number



Inheritance of Type and Code

- Distinguish between:
  - inheritance for code reuse (an implementation issue)
  - inheritance as a relationship between types (behavioural specification)

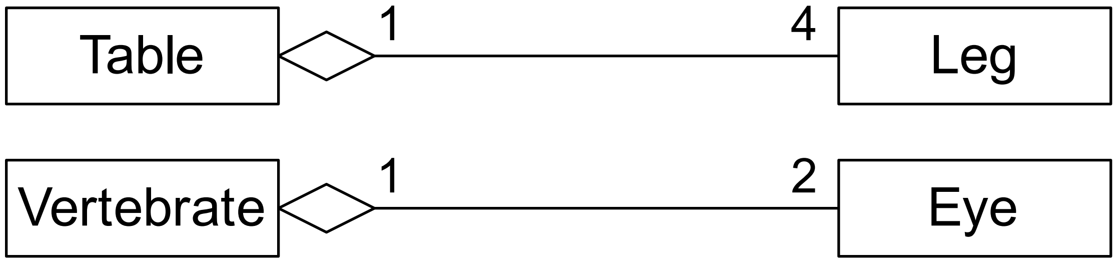


Using Classes

- O-O programming: designing and implementing classes.
- Problem split into hierarchies of classes.
- Classes — if mostly designed in advance.
- One check: if it is a subclass then there an “is-a” relationship:
  - Orange is-a Fruit (subclass)
  - Orange is *not* an Apple (≠ subclass)

Part-Whole Relations

- Assemblies and sub-assemblies or Aggregations
  - “is-part-of” relationship
- Down such hierarchy: inheritance of attributes
  - a table’s legs inherits its colour.
- Up: parts provide abilities to the whole.
  - we see because we have eyes.
  - called Delegation: object delegates responsibility to another



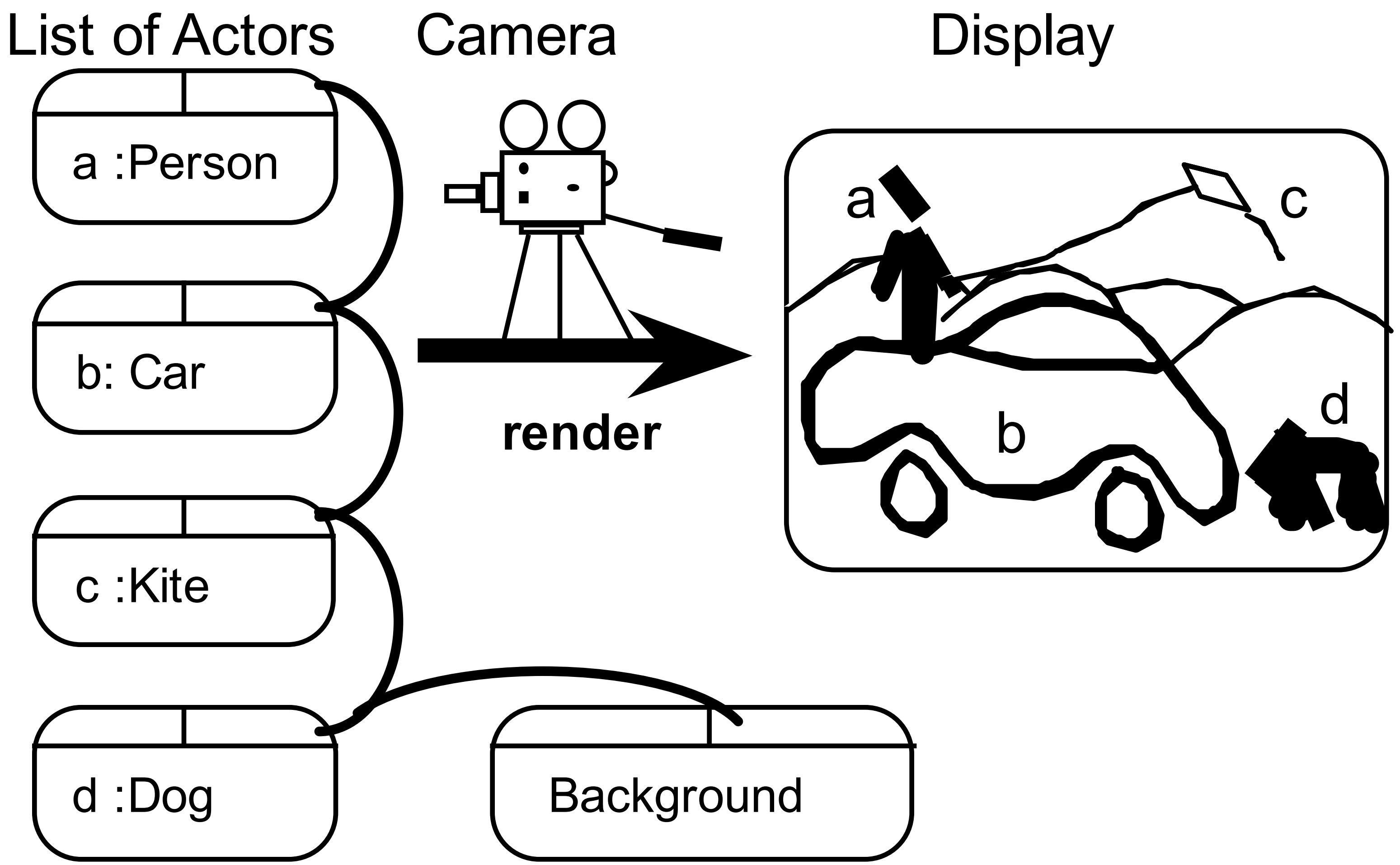
Polymorphism

- Same message
  sent to different classes
  with any type of argument.
- Class hierarchies ⇒ certain polymorphism
  - related but distinct classes understand the same messages
  - Inheritance allows automatic but controlled polymorphism
  - adding new types easy.

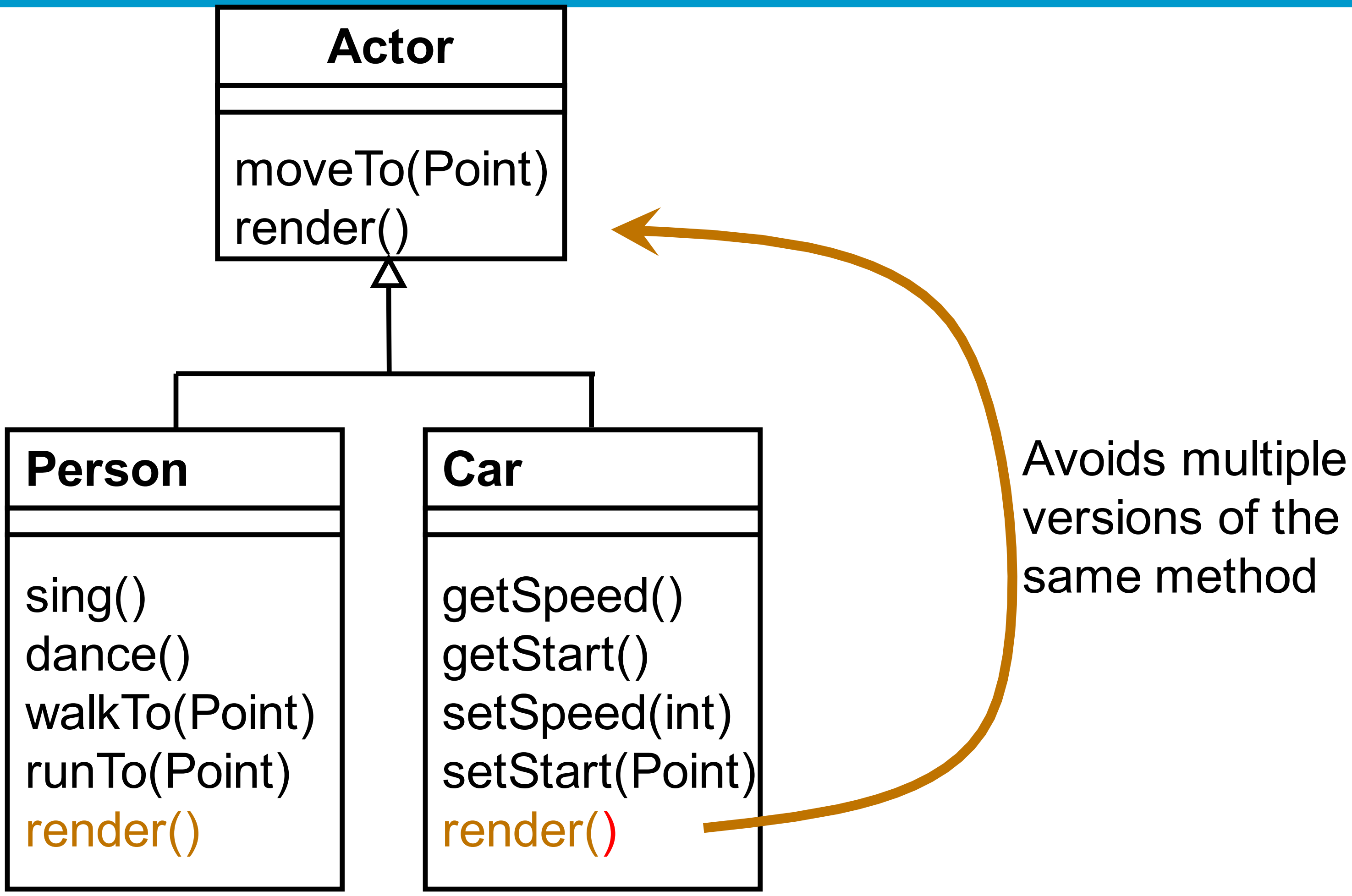
Polymorphism with Classes

2 + 3	“Send the message '+' to the SmallInteger 2 with the parameter 3”
5 * 3.5	“Send '*' to the SmallInteger 5 with the Float parameter 3.5”
2.7 + (1/3)	“Send '+' to the Float 2.7 with the Fraction parameter (1/3)”

Polymorphic message to render different kinds of Actors



Polymorphism and an Abstract Superclass



Summary of O-O Features.

- Objects.
- Message Passing.
- Classes.
- Inheritance.
- Interfaces: Information Hiding.
- Polymorphism.

ANALYSIS AND USE CASES

Project Deliverables Discussed Here

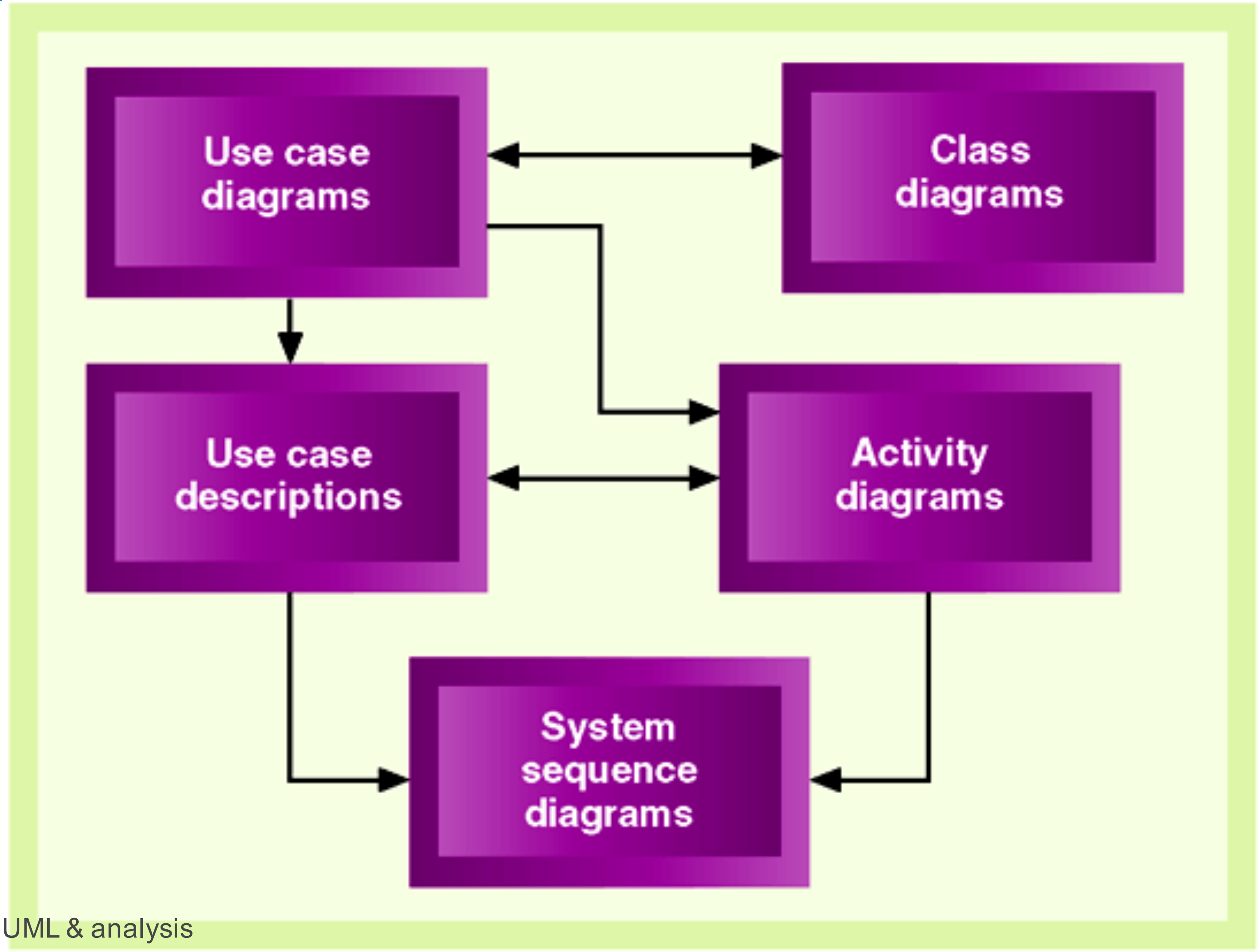
- Use case narratives
- Use case diagrams
- Analysis model
  - Class diagram
  - Object behavioural model (sequence and state diagrams) or
  - State Transition Diagram
- Project plan
- Test plan

Analysis Principles

- Begin by focusing on the essence of the problem without regard to implementation details
  - Examine “What” rather than “how”
- Understand the problem before you begin to create the analysis model.
  - Record the origin of and the reason for every requirement.
  - Use multiple views of requirements.
- Prioritize requirements.
- Develop (paper) prototypes that enable a user to understand how human-machine interaction will occur.

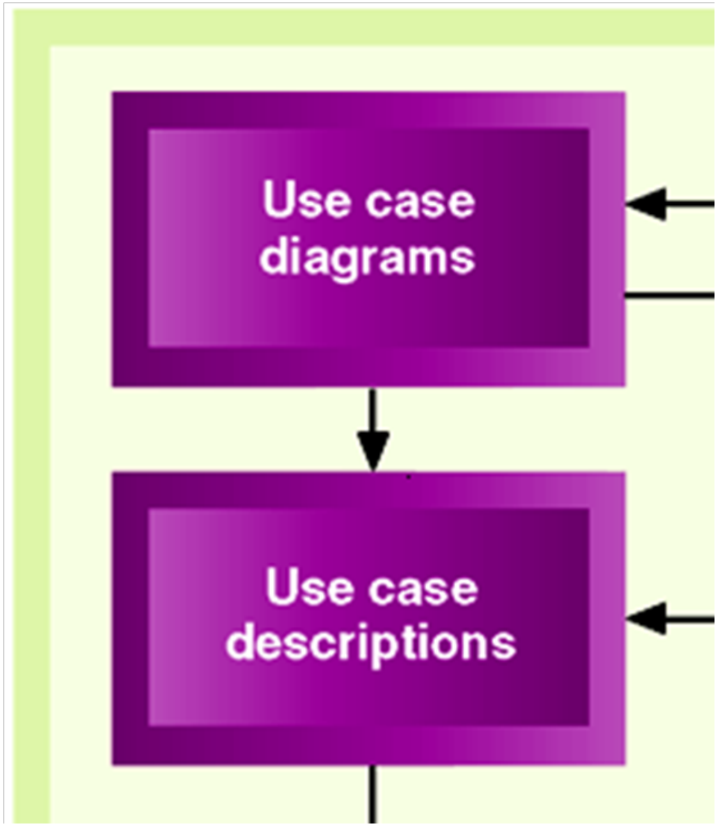
Work to eliminate ambiguity.

Analysis Model



Use Cases

- Use cases are one way to capture (especially) functional requirements.
  - They are stories of using a system.
- Issues:
  - Identify different use case levels
  - Write use cases in standard format
  - Contrast essential and concrete use cases
  - Apply guidelines
  - Read and write use case diagrams



Definition: Use Case

- Informally, a use case is a story of using a system to fulfil a goal.
  - Rent Videos
- Used by primary actors
  - E.g., shop assistant
  - External agents
  - something with behaviour
- Use supporting actors.
  - CreditAuthorizationSystem

The better use cases are simple and familiar because it makes it easier – especially for customers and end-users – to contribute to their definition or evaluation (Craig Larman)

Example: Brief Format Use Case

- Rent Videos.
  A Customer arrives with videos to rent. The Assistant enters their ID, and each video ID. The System outputs information on each. The Assistant requests the rental report. The System outputs it, which is given to the Customer with their videos.

Example: Alternative Use Case Narratives

For each use-case provide a narrative document

Use Case:	*Buy Item*
Actors:	*Customer, Cashier*

Description:

1. The use case begins when the customer arrives at a checkout with items to purchase.
2. The Cashier records each item. If there is more than one of an item, the Cashier can enter the quantity as well.
3. The system determines the item price and adds the 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 that item entry is complete.
5. The system calculates and presents the sale total.

Definition: Scenario

- Informally, a scenario is a specific sequence of actions and interactions in a use case.
  - One path through the use case.
  - E.g., The scenario of renting videos and first having to pay overdue fines.
  - Also known as use case instance
- More formally, a use case is a collection of success and failure scenarios describing a primary actor using a system to support a goal.
  - Use case can contains multiple scenarios (i.e., >1 path thru use case)

Definition: Actor

- Actor is anything with behaviour that triggers use case execution
  - Primary Actor accomplishes goal via use case
    - Typically a human user, also concepts like Time
  - Secondary Actor provides a service to use case
    - 3rd party system
  - Offstage actor is interested in use case results
    - Regulatory agency

Use Cases ≠ O-O

- There is nothing object-oriented about use cases.
- So, why bother?
  - We need some kind of requirements input for the design steps.
  - They are common/popular.

Levels of Use Cases

- A common challenge is identifying use cases at a useful goal level.
- For example, how do we know which of these is at a useful level?
  - Negotiate a Supplier Contract
  - Rent Videos
  - Log In
  - Start Up

Levels of Use Cases

- One answer is that they are all use cases.
  - Not helpful…
  - We can end up with too many fine-grain use cases
    - management and complexity problems.
  - Or “fat” use cases which span an entire organization.

Guidelines: EBP for Use Case Levels

- The Elementary Business Process (EBP) guideline.
- Focus on use cases at the level of EBPs.
  - “*A task performed by one person in one place at one time, in response to a business event, which adds measurable business value and leaves the data in a consistent state.*”
    - Approve credit order — OK.
    - Negotiate a supplier contract — not OK.

Use Case Levels: Applying the Guidelines

- Applying the EBP and size guidelines:
  - Negotiate a Supplier Contract
  - Rent Videos
  - Log In
  - Start Up
- The others can also be modelled as use cases.
  - But, prefer a focus on the EBP level.

Definition: Essential & Concrete Use Cases

- “Keep the UI out”
- Essential use cases defer the details of the UI, and focus on the intentions of the actors.
- Essential: Assistant enters Customer ID.
- Concrete/worse: Assistant takes Customer ID card and reads the bar code with laser scanner.

Guidelines: Use Case Writing I

- The name of a use case should start with a verb
- Start sentence 1 with “<Actor> does <event>”
  - Customer arrives with videos to rent.
- First write in the essential form, and “Keep the UI out.”
- Capitalize “actor” names.
  1. …
  2. Assistant enters…
  3. System outputs…

Guidelines: Use Case Writing II

- Terse is good. People don’t like reading requirements ;). Avoid noisy words.
- More verbose
  1. …
  2. The assistant enters…
  3. The system outputs…
- Less
  1. …
  2. *Assistant* enters…
  3. *System* outputs…

Guidelines: Use Case Writing III

- Use case steps are a dialogue between user and system, for example…
    1. User stimulates the system
    2. System responds to the user
    3. User stimulates the system again….
- Keep text simple and clear
    - Minimize articles (‘a’, ‘an’, ‘the’)
    - Maintain present tense
    - Always active voice
    - Ideally, 1 action per step

Motivation: Comprehensible & Familiar

- Use cases are stories.
- A simple and familiar model that many people, especially non-technical, can easily relate to.

Use Case Formality Types

Use in your project

- Brief – terse one-paragraph summary
  - Usually of main success story
  - Can be used as a first draft of use case description
  - Common in agile environments (user story / backlog item) where detailed requirements are often communicated face to face
- Intermediate/Casual – informal paragraph
  - Multiple paragraphs for various scenarios
  - Appropriate for high-level process or when stakeholders possess a good understanding of feature
- Fully Dressed – all steps written in detail
  - Detailed requirements definition approach
  - Pre and post conditions included

Examples: Brief format

- Can be brief - a short, single paragraph summary, usually of the main success scenario

Reserve a Flight online
Actor: Customer

The customer enters the dates and destination they wish to travel to. The system looks up the most suitable flight and routing. The customer selects the most appropriate route. The system calculates the price. The customer confirms the route and the system calculates the fare.

Examples: Intermediate/Casual

- Informal format – covers important alternative paths

Reserve a Flight online
Actor: Customer

The customer then specifies which airline they would like to travel on. They enter the dates and destination they wish to travel to. The system looks up the most suitable flight and routing. Where the proposed travel dates or destination are not feasible, the customer is requested to modify their travel plan.

The customer selects the most appropriate route. The system calculates the price and reminds the user that the reservation is only valid for the specified number of days. The system also updates the availability of flights for that date. The customer confirms the route and the system calculates the fare in the local currency of the customer. The customer can cancel or change the reservation at anytime within the number of days that the reservation is valid.

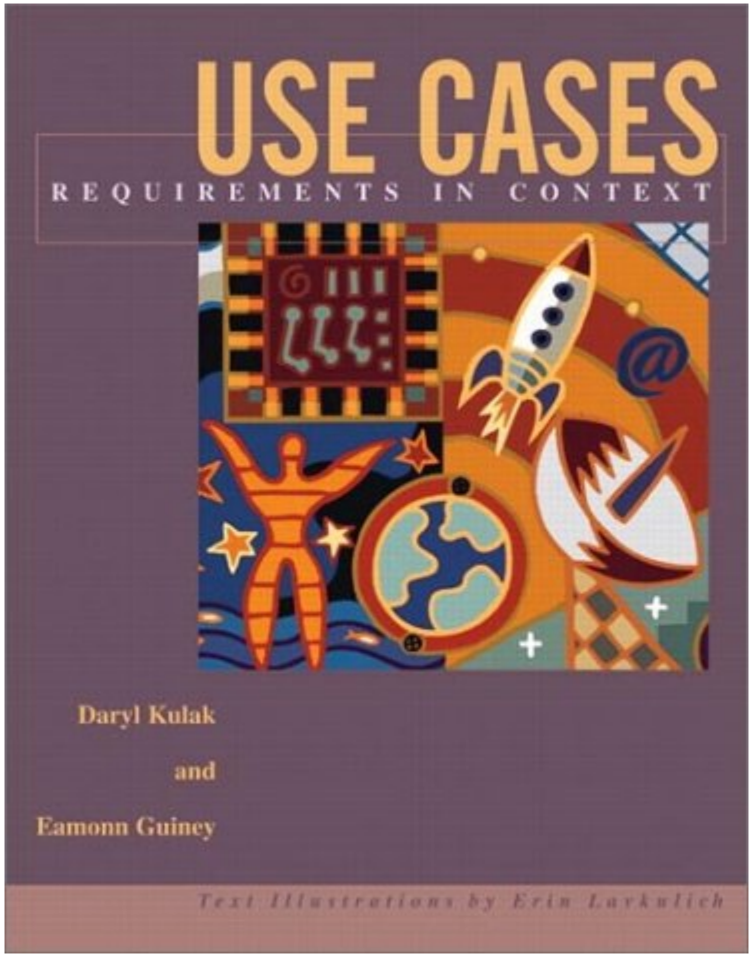
Examples: Fully Developed

Normally a standardised template with all steps and variations documented. Contains some or all of the following supporting sections:

- Use Case Name and number
- Level and Priority
- Brief Description
- Actors (Primary / Secondary)
- Stakeholders and Interests
- Related Use Cases
- Trigger
- Non Functional Requirements
- Preconditions
- Post Conditions (Guarantees)
- Flow of Events (main success scenario)
- Exception Conditions
- Business Rules
- Input / Output / User Interface information
- Frequency of Occurrence
- Open Issues
- Test Cases
- There are others………

Motivation: “Requirements in Context”

- Use cases bring together related requirements.
- More cohesion and context for related requirements.



Concrete Use Cases

- Sometime after the essential form of the use case has been written, one may optionally write it in a concrete form.
- Customer arrives at a checkout with videos or games to rent.

Assistant scans Customer ID…

Extensions

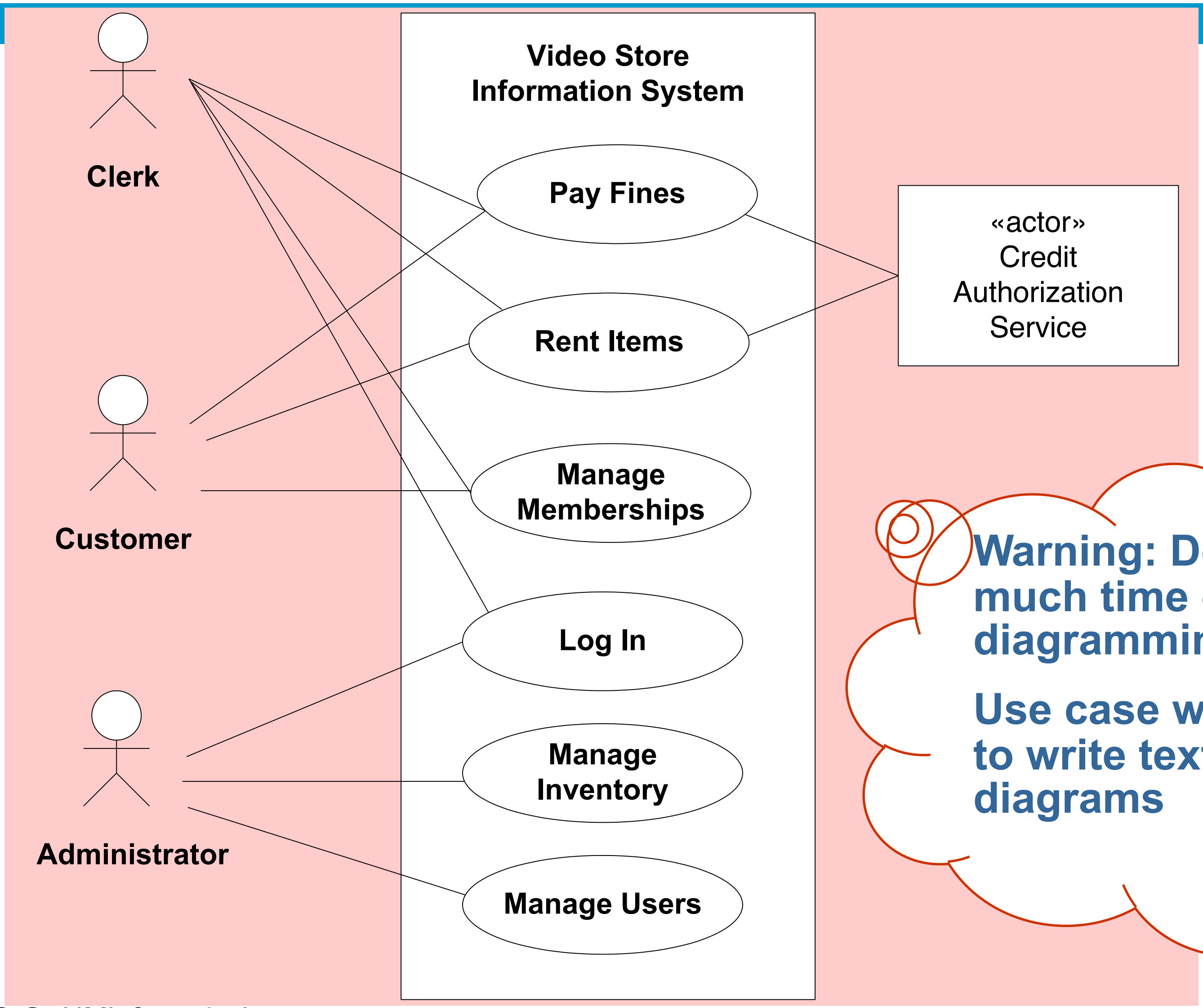
2a. Scanner failed.

 1. Assistant enters ID on keyboard (see GUI window example, fig 5)…

Use Case Diagrams

- The UML has use case diagrams.
- Use cases are *text*, not diagrams.
  - Use case analysis is a *writing* effort, not drawing.
- But a *short* time drawing a use case diagram provides a context for:
  - identifying use cases by name
  - creating a “context diagram”

Use Case Diagrams I

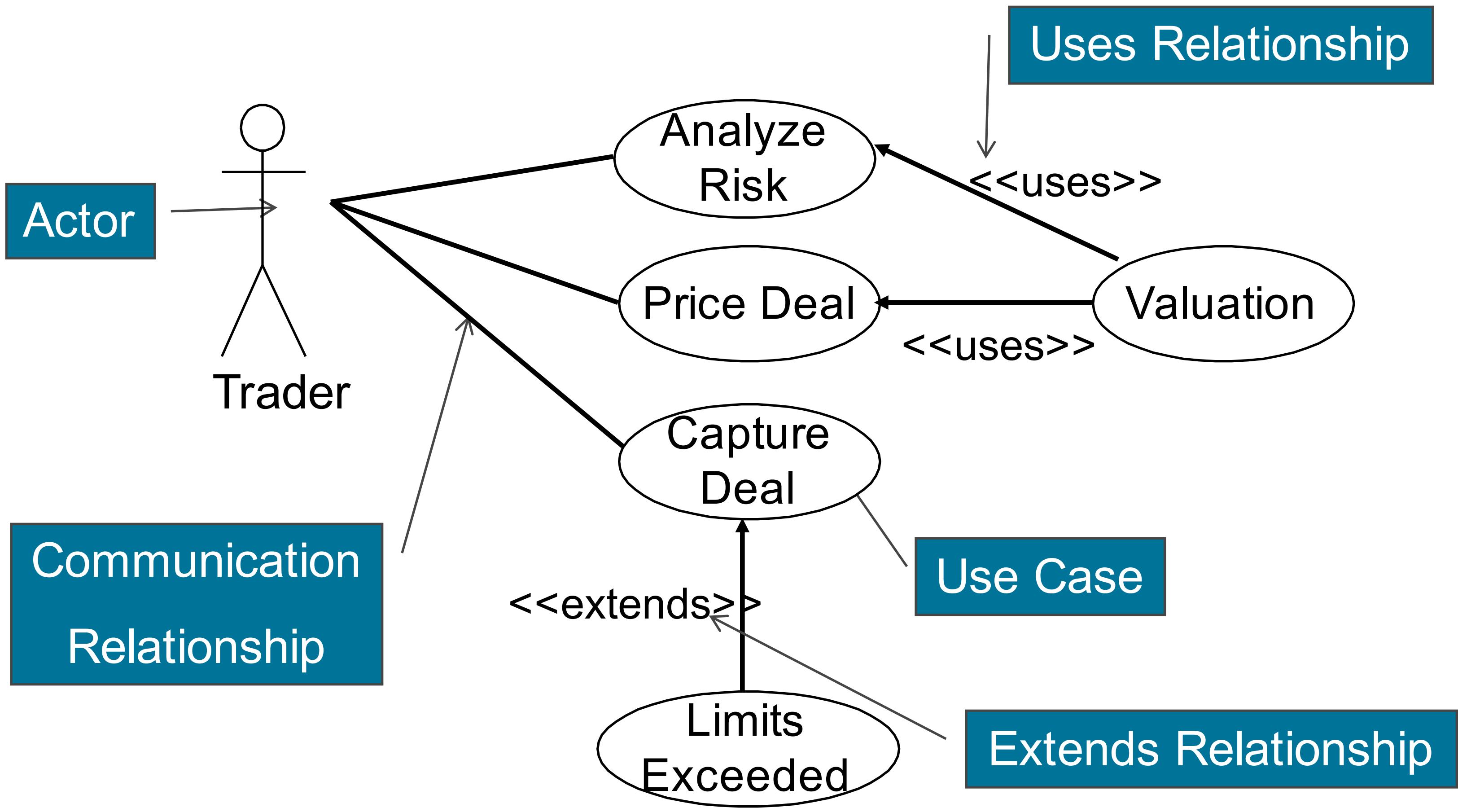


Warning: Don't spend much time on diagramming.

Use case work means to write text, not draw diagrams

Use Case Diagrams II

Graphically shows use-cases, actors and their relationships.



UML

UML

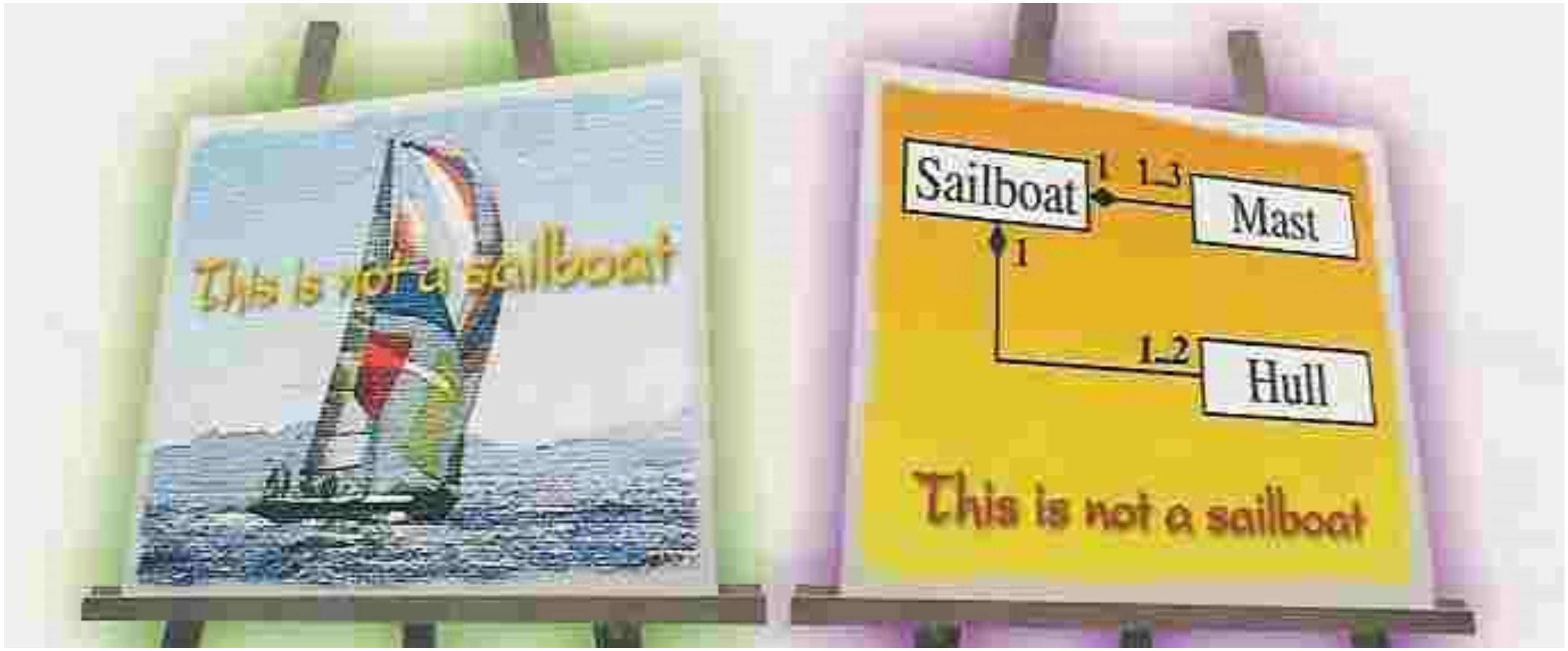
- Diagramming notation standard.
- Open method used to specify, visualize, construct and document the artefacts of an object-oriented software system
- Not a process, or design guide.
- The point is not to learn UML but to learn o-o design!

Aspects of Modelling in UML

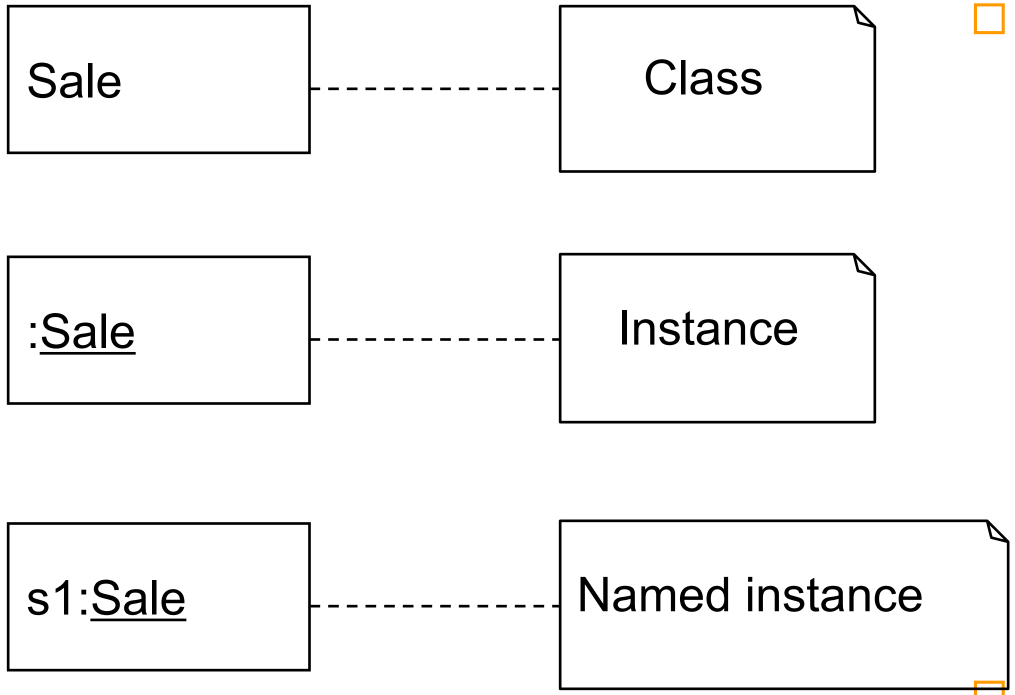
- Functional Model:
  - Functionality from the user's Point of View
    - Use Cases Diagrams.
- Object Model
  - System Structure: objects, attributes, operations, & associations
    - Class Diagrams.
- Dynamic Model
  - Internal behaviour of the system.
    - Sequence Diagrams
    - Activity Diagrams
    - Statechart Diagrams.

A Diagram is Not a Model

- a UML diagram is a graphical representation of the information in the model
- the model exists independently



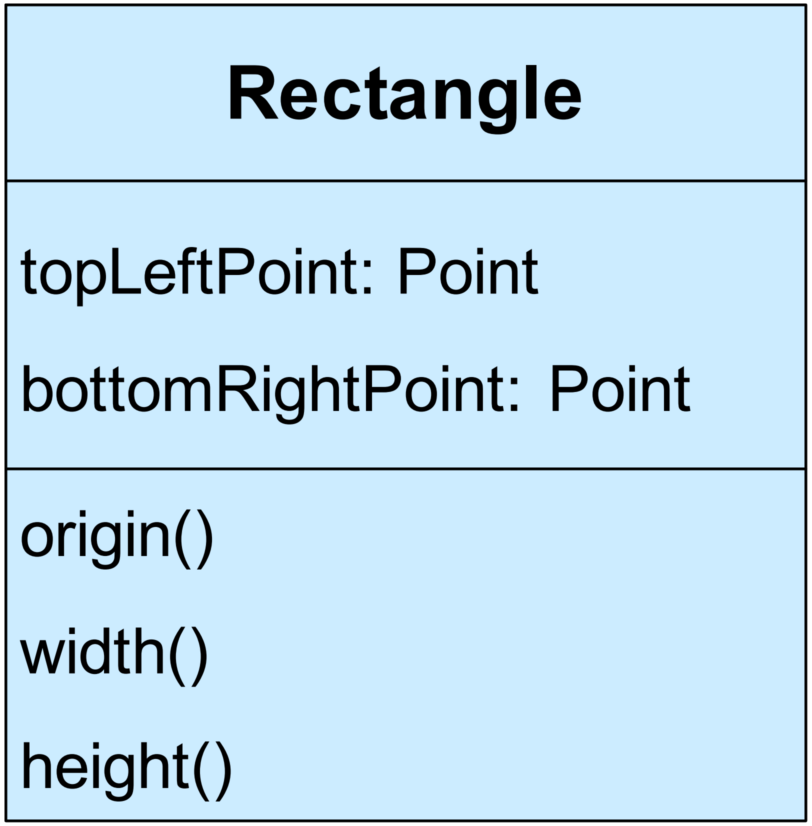
UML: Illustrating Classes and Instances



- To show an instance of a class, the regular class box graphic symbol is used
  - The name is underlined if writing (for extra emphasis).
  - Additionally a class name should be preceded by a colon.
- An instance name can be used to uniquely identify the instance.

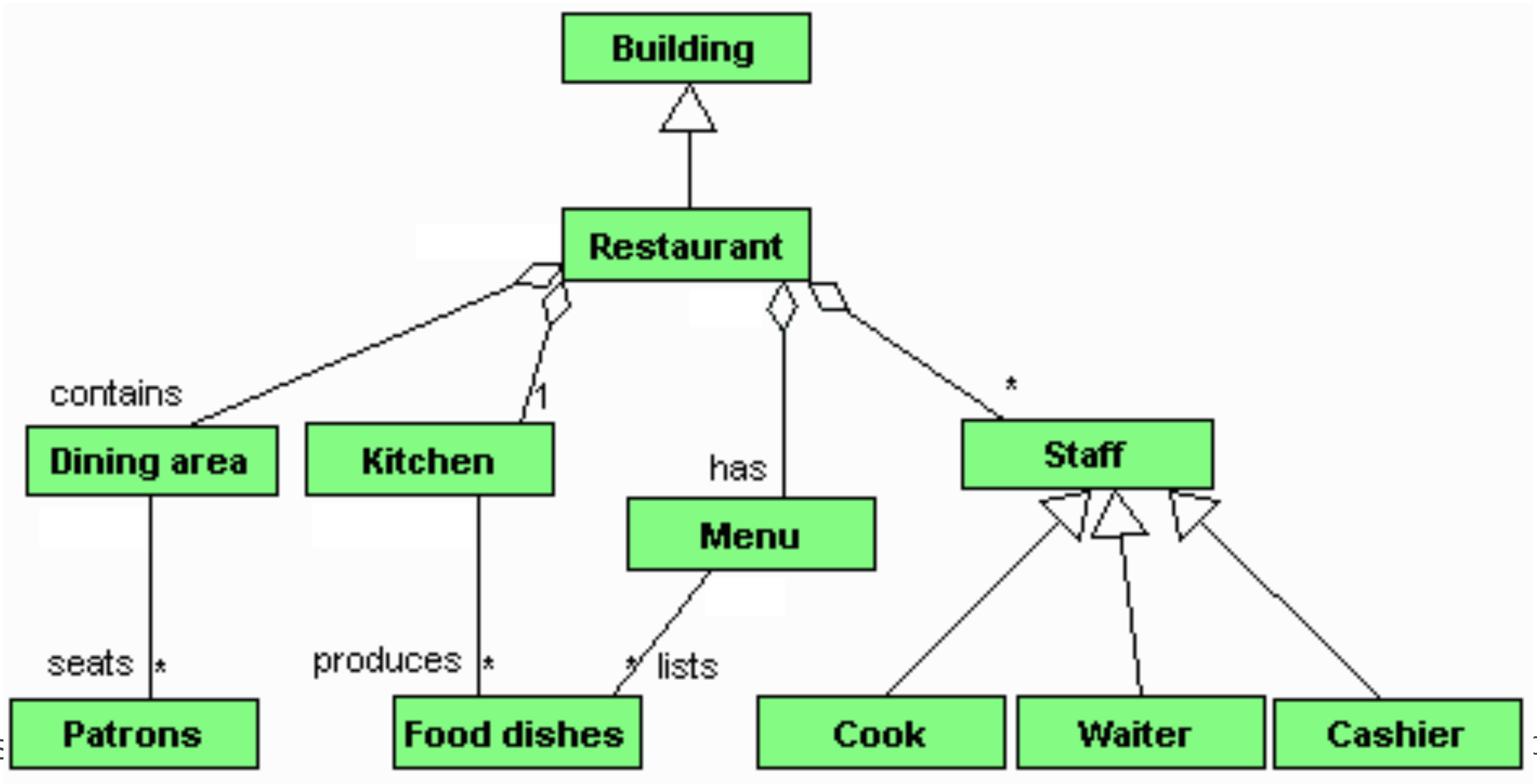
Rectangle Class Diagram

- Top: Name of class
- Middle: Attributes — usually fields (instance variables)
  - type is optional
- Bottom: Operations — usually methods



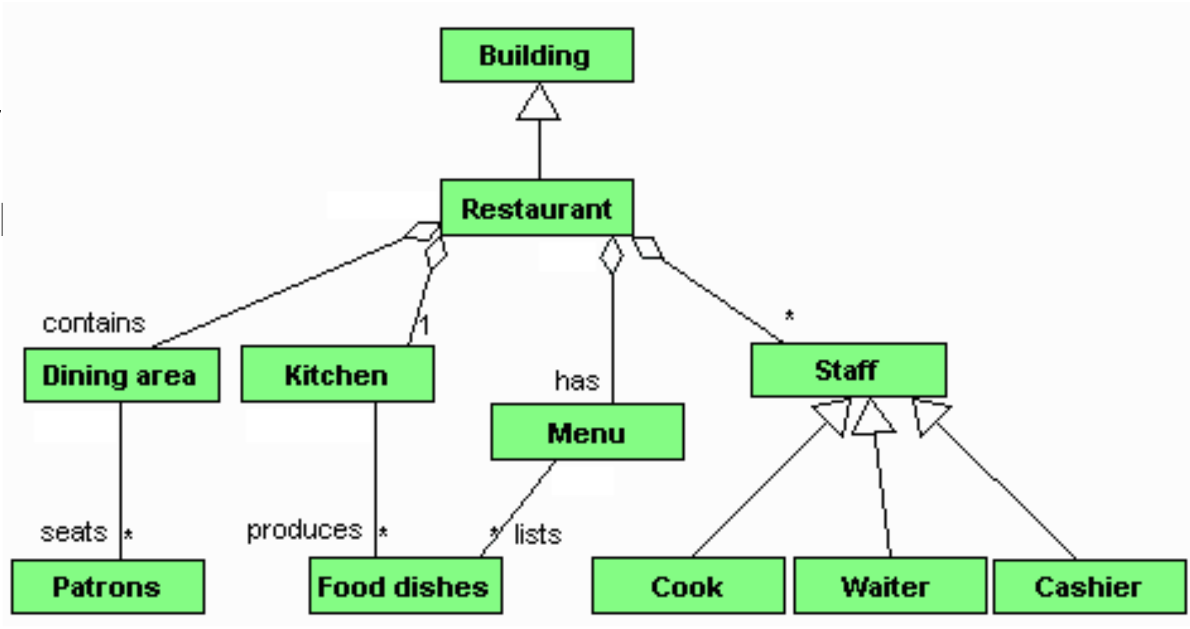
UML Class Diagram

- Arrow with triangular hollow head points to superclass
- Hollow diamond indicates part/whole relationship



Simple Restaurant UML Class Diagram

- Restaurant is a building.
- Restaurant contains a dining area
  - ...which seats zero or many Patrons.
- Restaurant has exactly one Kitchen
  - ...which produces zero or many Food dishes.
- Restaurant has Menu.
- Menus list zero or mor
- Restaurant has zero o
  - Cook is a Staff.
  - Waiter is a Staff.
  - Cashier is a Staff.



UML Conventions: Restaurant System

- Inheritance relationships with a triangle
- Containers with diamond shape.
- The role of the relationship may be specified as well as the cardinality.
  - any number of Food dishes (*)
  - one Kitchen (1)
  - Dining Area (contains)
  - any number of staff (*)
  - All of these objects are associated to one Restaurant.

UML: What's Important?

Harmful is knowing how to read and draw UML diagrams, but not being an expert in design and patterns.

Important is object and architectural design skills, not UML diagrams, drawing, or CASE tools.

DYNAMIC BEHAVIOUR VIEWS

Introduction to Object Oriented Development

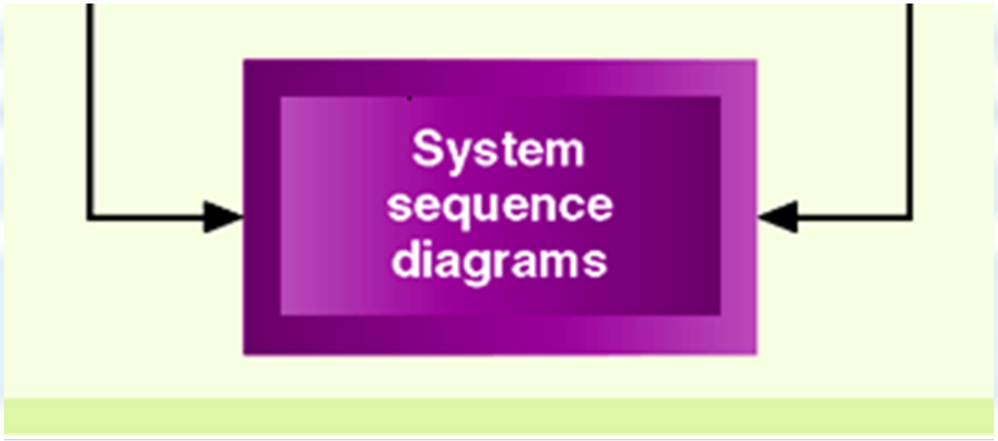
Inception Phase: Use Cases and Analysis

UML

Dynamic Behaviour Views

- Interaction Diagrams
- Communication Diagrams

Structural Views



Object-Behavioural Modelling

- Class models are static
- Object-behaviour model ➔ dynamic (function of specific events and time)
- Process:
  1. Evaluate use-cases to understand the sequence of system interaction
  2. Identify events that drive the interaction sequence and relate these to specific objects
  3. Create an interaction diagram for each use-case
  4. Build a state diagram for the system
  5. Review model to verify accuracy and consistency

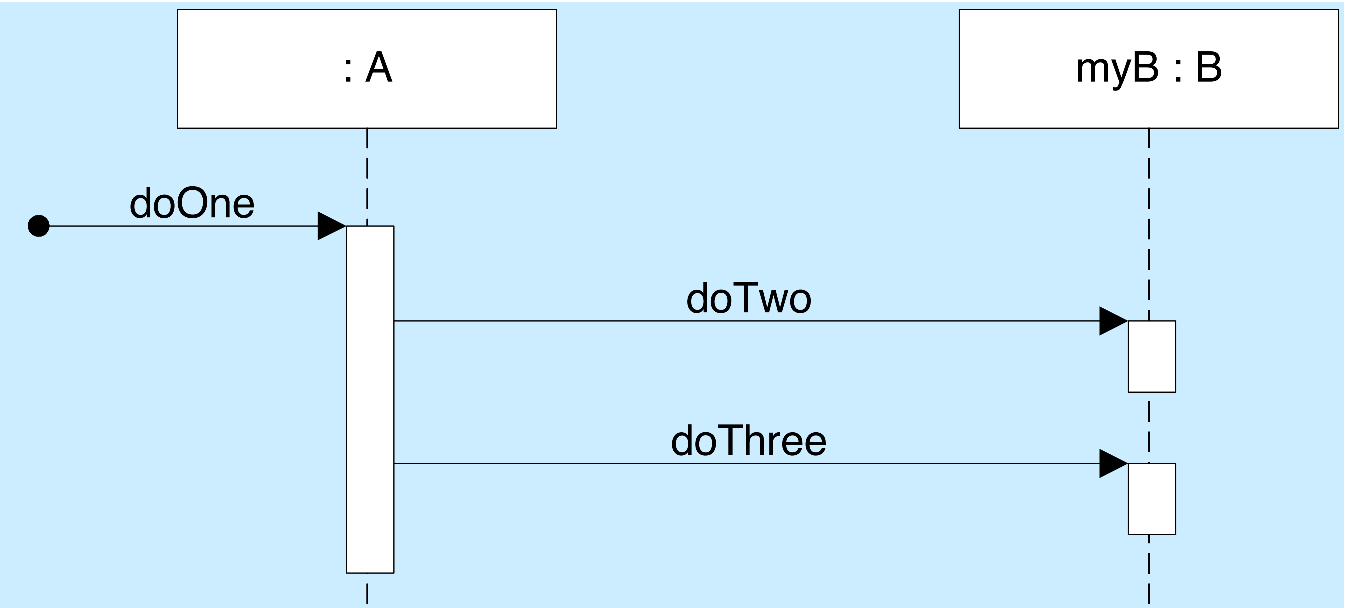
Interaction Diagrams

- Sequence *vs*. Communication diagrams
  - Sequence Diagrams
    - Easier to see sequence of method calls over time
    - More expressive UML notation
    - Better support from many tools
  - Communication Diagrams
    - Better fit on limited space (page or whiteboard)
    - Easier to edit/amend
    - Free form
      - Messages have to be labelled in chronological order

Sequence Diagram and Code

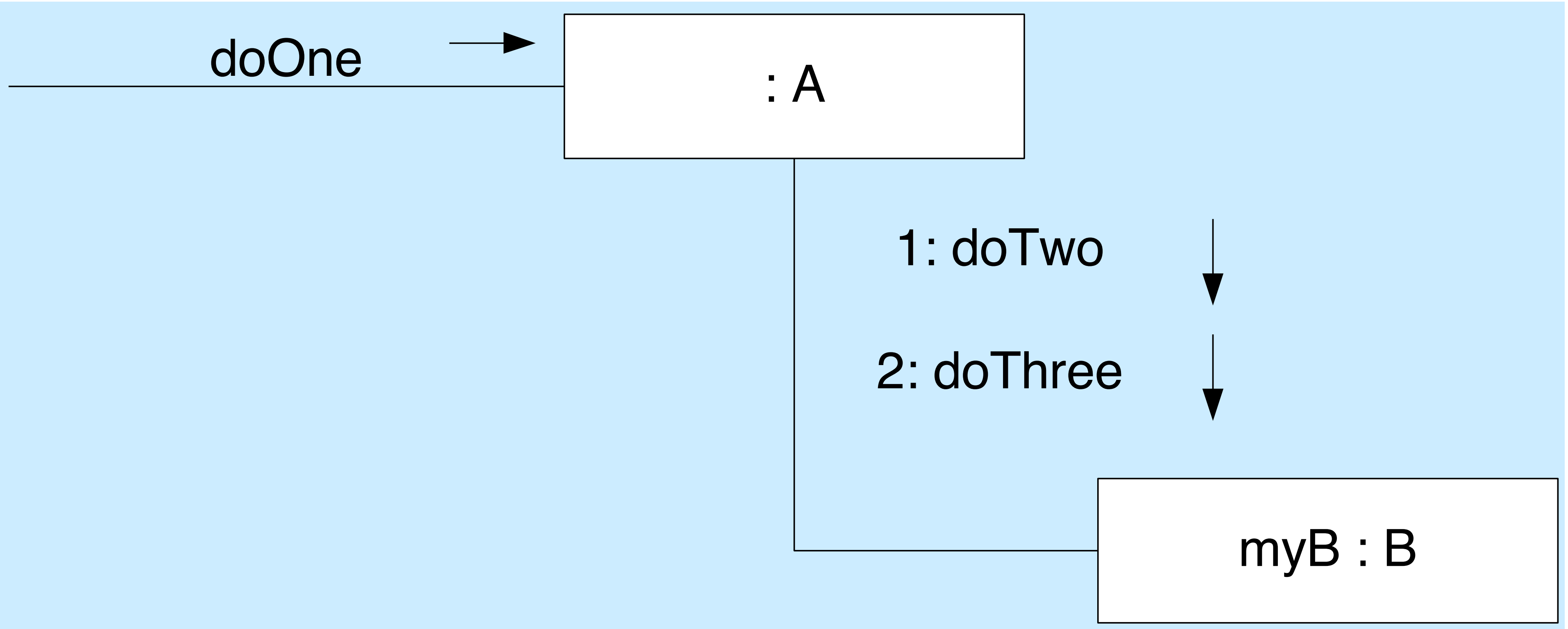
```
public class A {
  private B myB = new B();
  public void doOne() {
       myB.doTwo();
       myB.doThree();
  }
}
```

- Shows *dynamic* communications
- Shows methods calls with horizontal arrows from caller to called.
- Dotted line shows object's lifeline.
- White bar shows activation (lifetime of method) on object's lifeline.

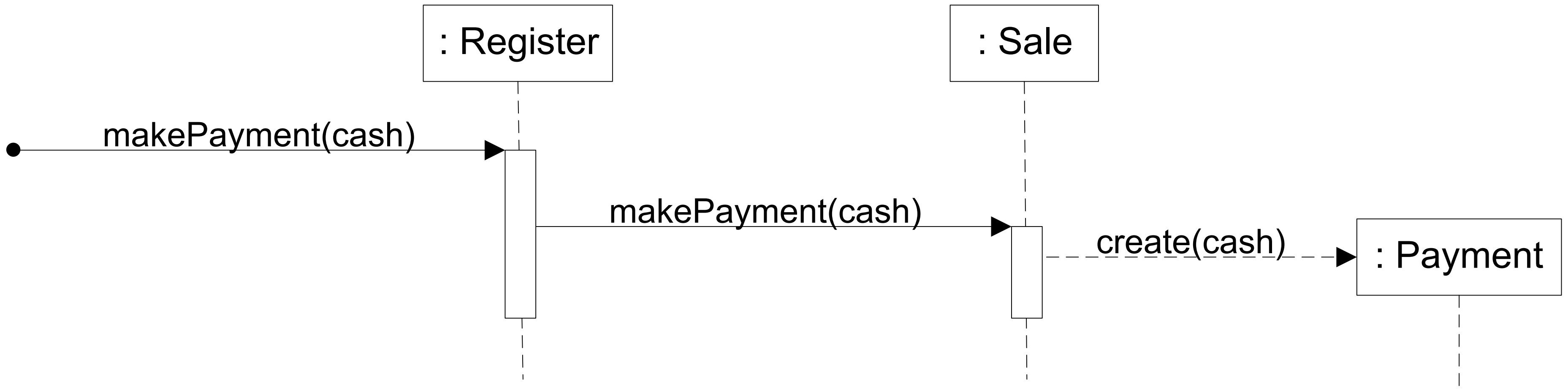


Communication Diagram

- Same collaboration using communication diagram
  - Uses network (or graph) format
  - Objects can be placed anywhere



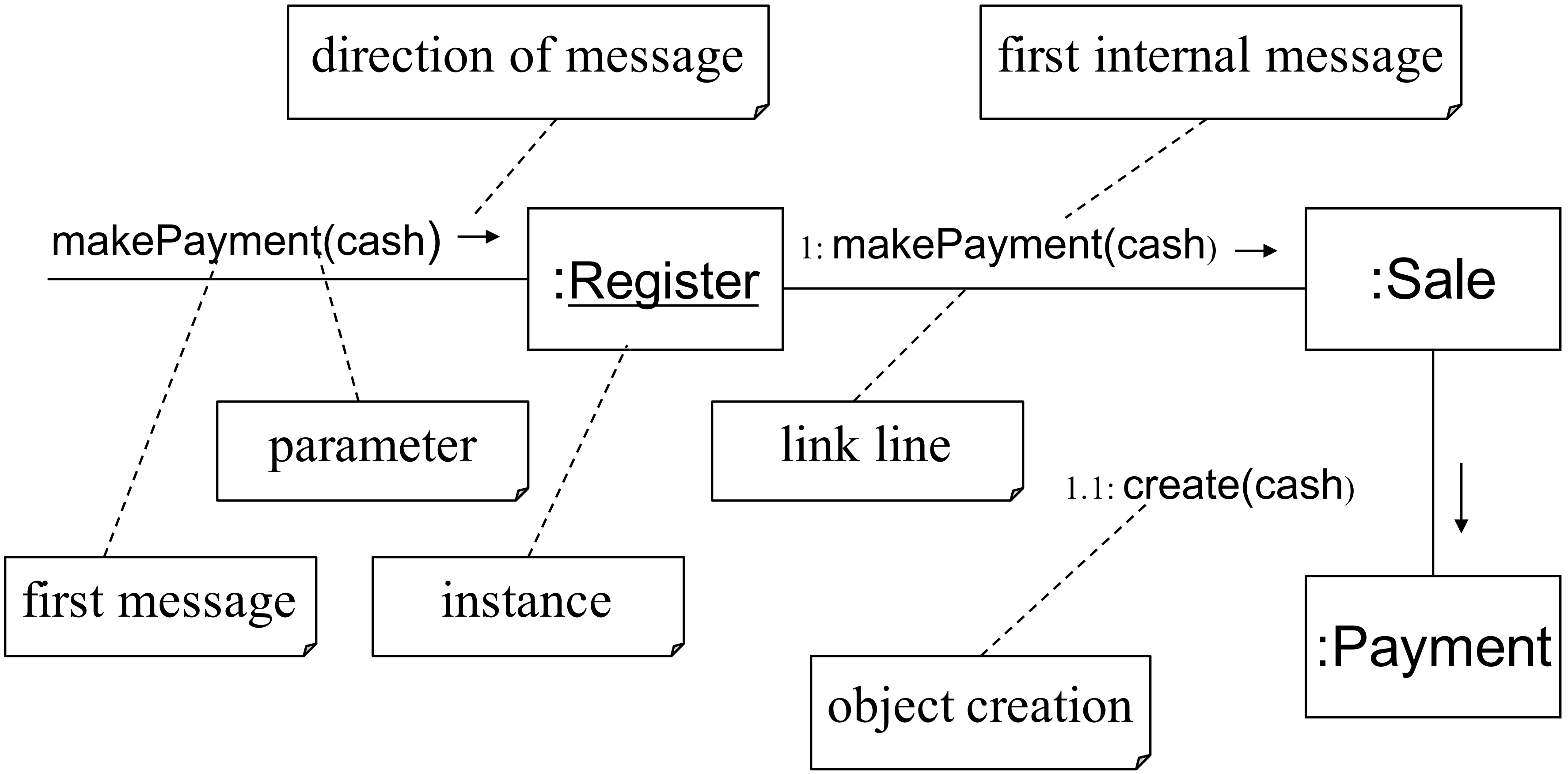
Example Sequence Diagram: makePayment



1. Someone(?) sends makePayment(..) message to Register
2. Register sends makePayment(..) message to Sale
3. Sale creates an instance of Payment

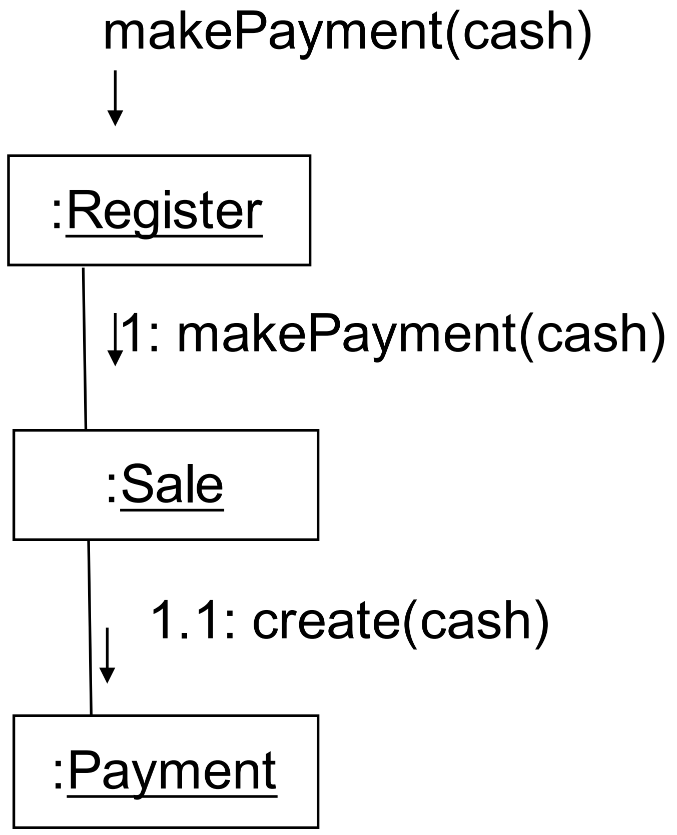
- Who created Register & Sale?
  - Interaction Diagrams show a fragment of system behaviour during isolated snapshot in time.
  - This can be confusing and lead to modelling errors/omissions!

Example Communication Diagram: makePayment

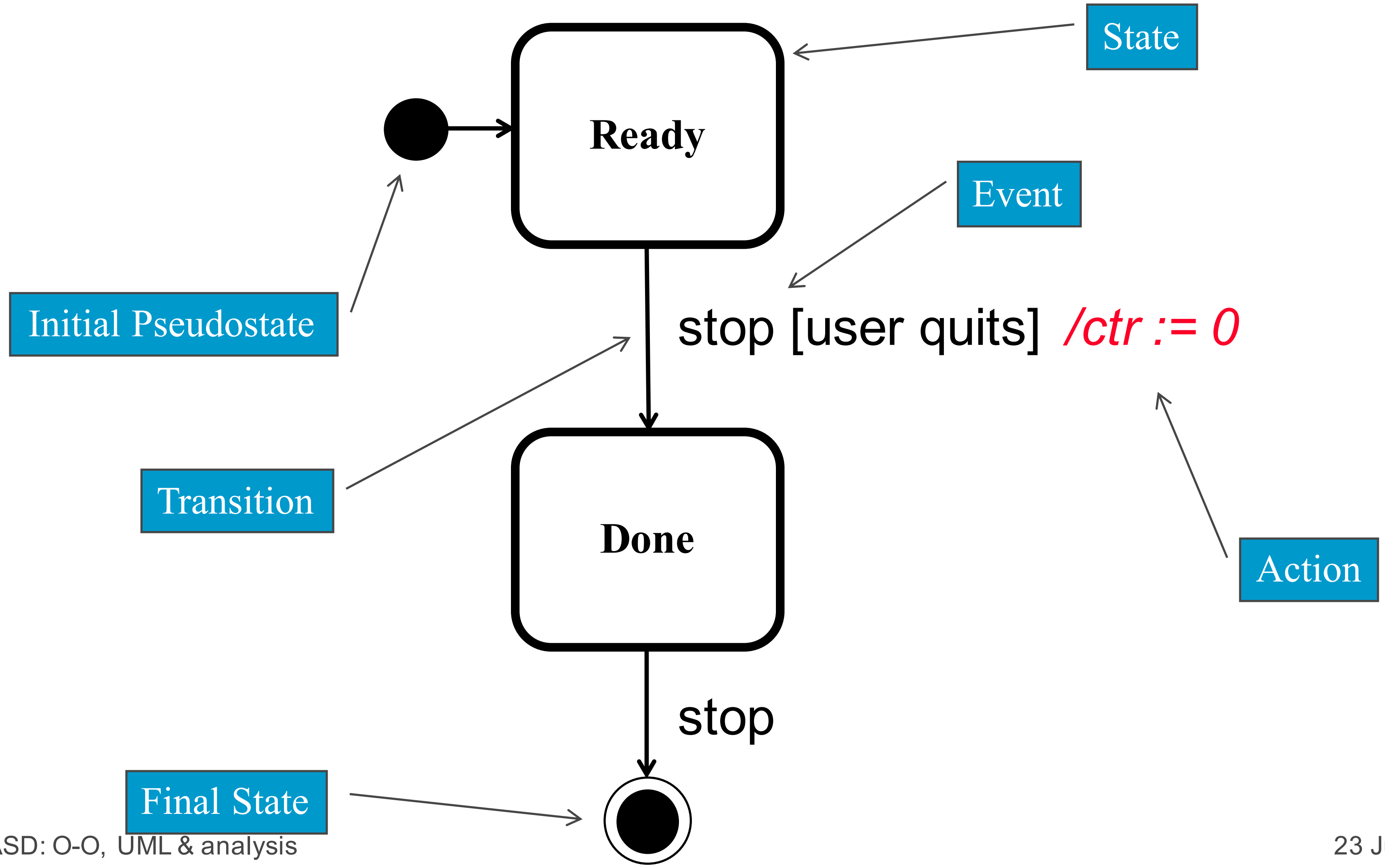


How to Read the makePayment Communication Diagram

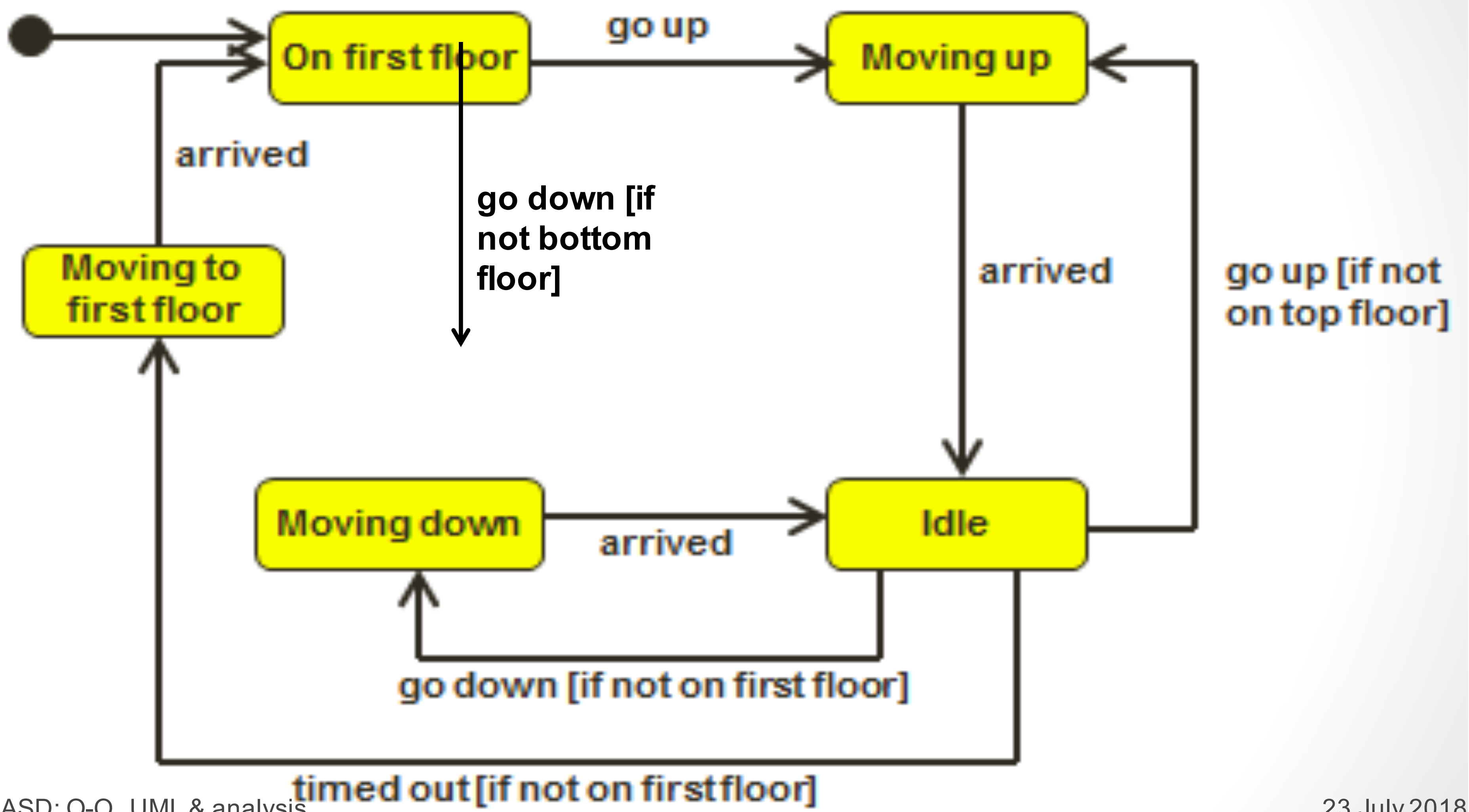
1. The message makePayment is sent to an instance of Register. The sender is not identified.
2. The Register instance sends the makePayment message to a Sale instance.
3. The Sale instance creates an instance of a Payment.



Basic UML State Diagram



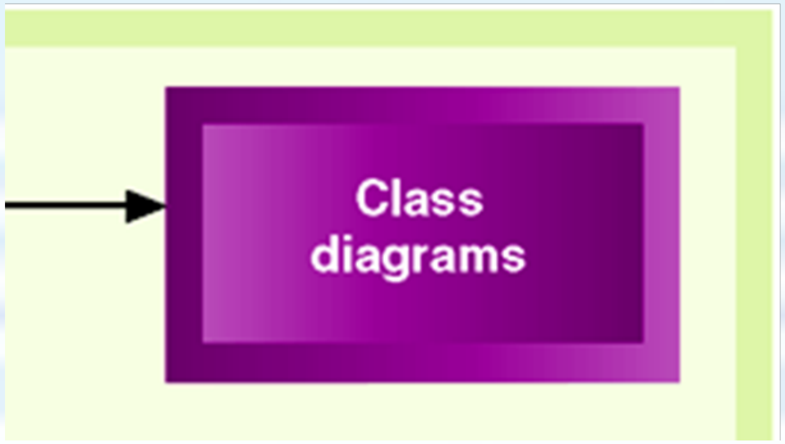
State machine for a lift


On first floor
go up
Moving up
arrived
go down [if not bottom floor]
arrived
go up [if not on top floor]
Moving to first floor
Moving down
arrived
Idle
go down [if not on first floor]
timed out [if not on first floor]

Essential UML models for OOAD

- Use cases
  - Functional requirements
- Class diagram
  - Objects with knowledge (attributes) and behaviour (operations)
  - Static relationships between objects
- Interaction diagrams: Essential dynamic view
  - Dynamic collaboration between objects
  - Don’t have to use full syntax in all cases

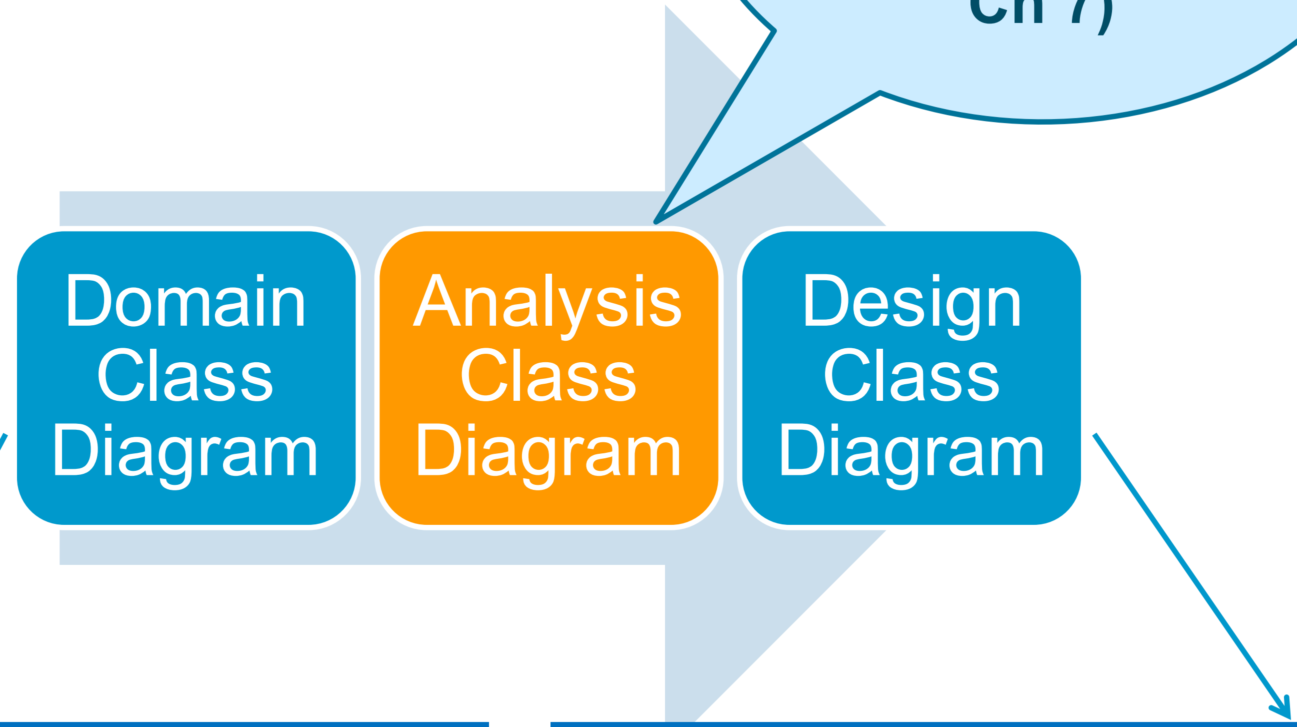
STRUCTURAL VIEWS



Class Diagram — Purpose

Identifies and documents the objects in the system

Use in your Stage 2 Deliverable (see Bennett Ch 7)



Visual dictionary of the noteworthy abstractions, domain vocabulary and information content of the domain

Completed it represents a Design Class Diagram - a detailed and rigorous design specification of the objects in the system

Class Diagrams — Summary

- **Domain Class** diagrams assist in identifying the important business concepts and rules around their relationships in the application domain
- **Analysis Class** diagrams are a refinement to the domain class diagram and represent the things in the user domain as software rather than concepts
- **Walkthroughs** where the Analysis Class diagram is interrogated to ensure it can support the detailed user requirements (as described in the use case descriptions) are an important quality assurance technique

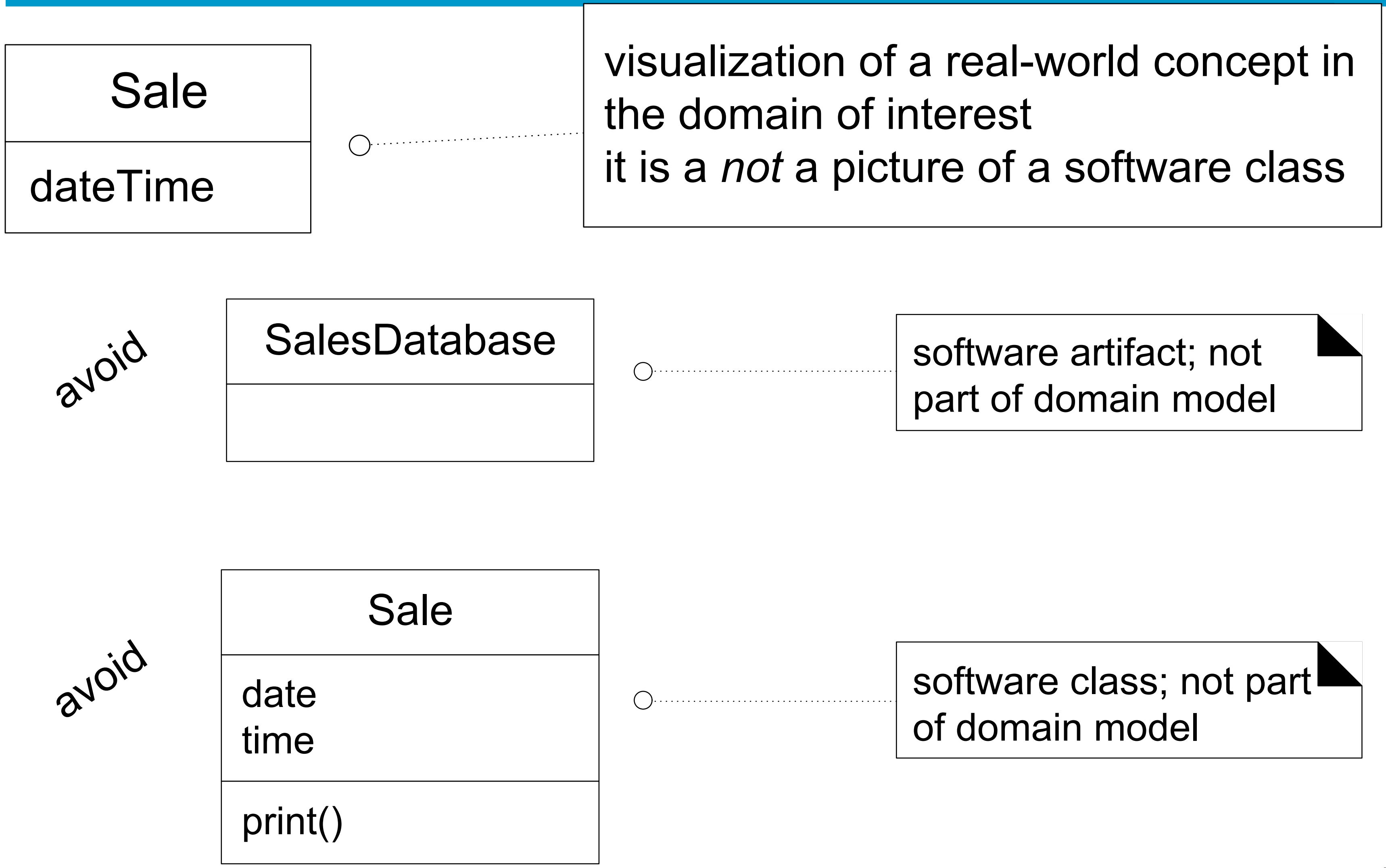
Class Diagrams

- Show the static structure of the model
  - the entities that exist
  - internal structure
  - relationship to other entities
- Static View
  - Do not show temporal information
- Used in both analysis (conceptual) and design (specification and implementation).
- Conceptual class diagrams only require a subset of the full UML notation.

Domain Models

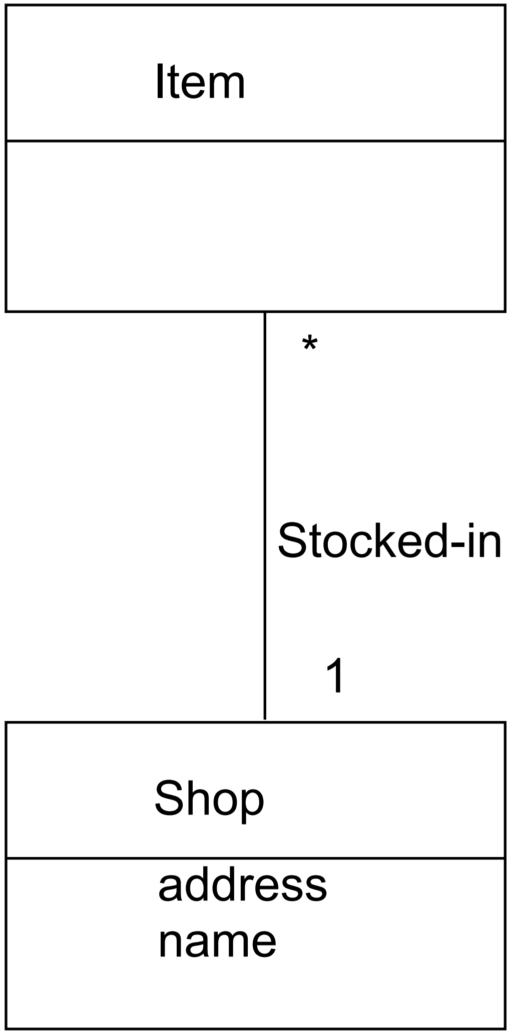
- A *Domain Model* visualizes, using UML class diagram notation, noteworthy concepts or objects.
  - It is a kind of visual dictionary
  - *Not* a picture of software classes
- It helps us identify, relate and visualize important information
- It is a set of static structure diagrams; no operations are defined
- It may show:
  - concepts
  - associations between concepts
  - attributes of concepts

Conceptual Classes *not* Software Classes

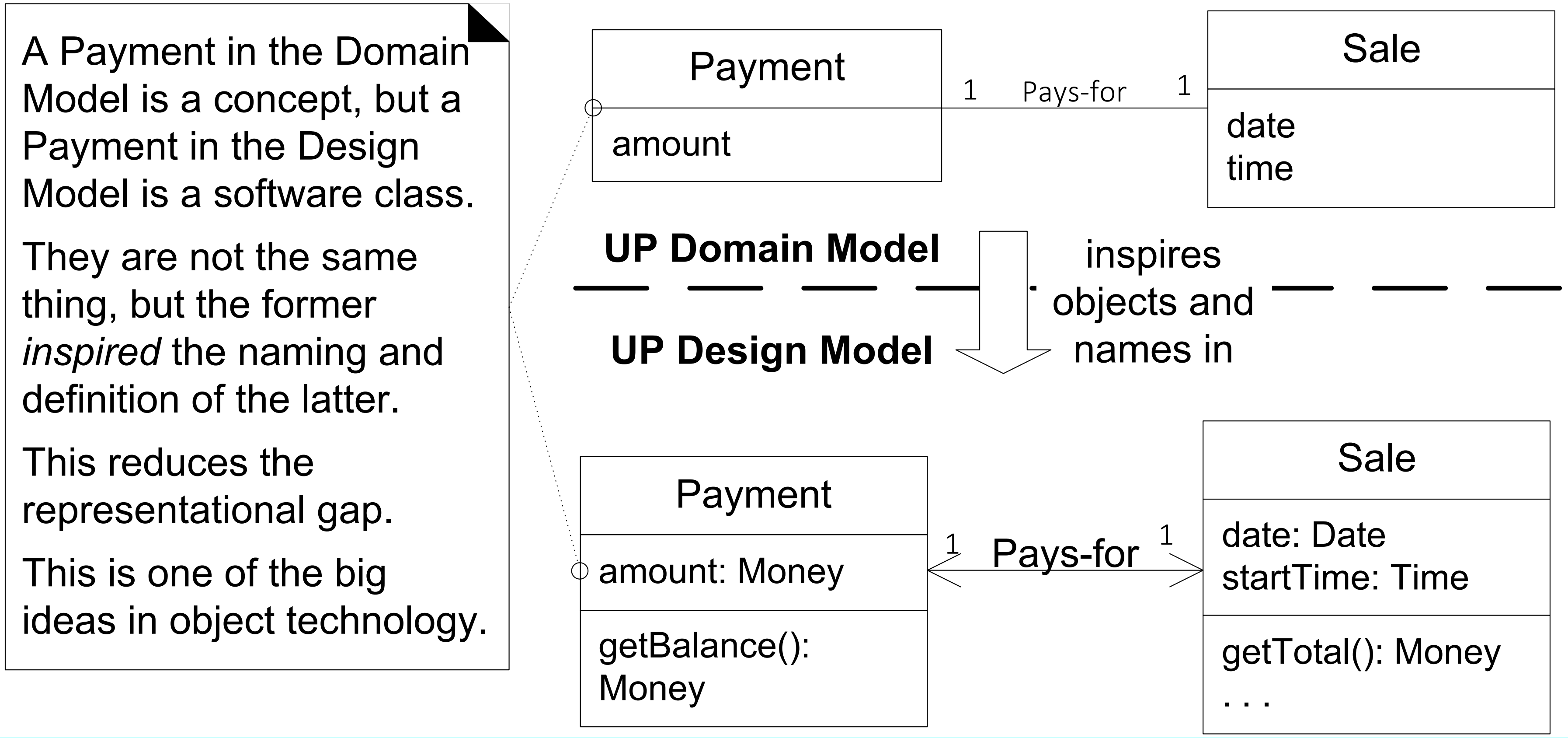

Sale
dateTime
visualization of a real-world concept in the domain of interest
it is a *not* a picture of a software class
avoid
SalesDatabase
software artifact; not part of domain model
avoid
Sale
date
time
print()
software class; not part of domain model

Domain Models

- A Domain Model is a description of concepts in the real world.
  - not software components.
- A concept is an idea, thing, or object.
- *Inspiration* for later creation of software design classes ⇒ reduce “representational gap.”



RUP Domain Model: Stakeholder's view of the noteworthy concepts in the domain


A Payment in the Domain Model is a concept, but a Payment in the Design Model is a software class.
They are not the same thing, but the former *inspired* the naming and definition of the latter.
This reduces the representational gap.
This is one of the big ideas in object technology.
Payment
amount
1
Pays-for
1
Sale
date
time
UP Domain Model
inspires objects and names in
UP Design Model
Payment
amount: Money
getBalance(): Money
1
Pays-for
1
Sale
date: Date
startTime: Time
getTotal(): Money
. . .

- The object-oriented developer has taken inspiration from the real world domain in creating software classes.
- Therefore, the representational gap between how stakeholders conceive the domain, and its representation in software, has been lowered.

Strategies to Identify Conceptual Classes

- Use a conceptual class category list.
  - Make a list of candidate concepts.
- 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

Concept Category	Example
Physical or tangible objects	Item, Register, Board
Descriptions of things	ProductDescription
Place of service	Store
Business transactions	Sale, Payment
Transaction line items	SalesLineItem
Roles of people	Cashier, Customer
Containers of other things	Store, Bin

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.**
4. …

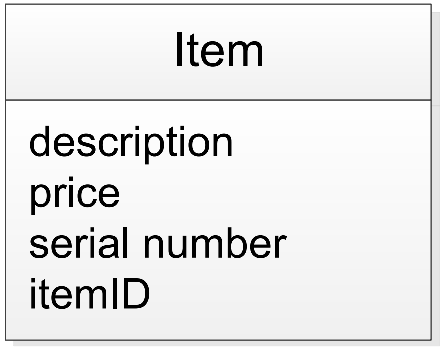
- The fully addressed Use Cases are an excellent description to draw for this analysis.
- Some of these noun phrases are candidate concepts; some may be attributes of concepts.
- A mechanical noun-to-concept mapping is not possible, as words in a natural language are (sometimes) ambiguous.

Objects or Attributes?

- Attributes are ‘simple’ data types
  - e.g., number, text
- Concepts that are *described* by simple attributes are objects
  - A shop has an address, phone number, etc.

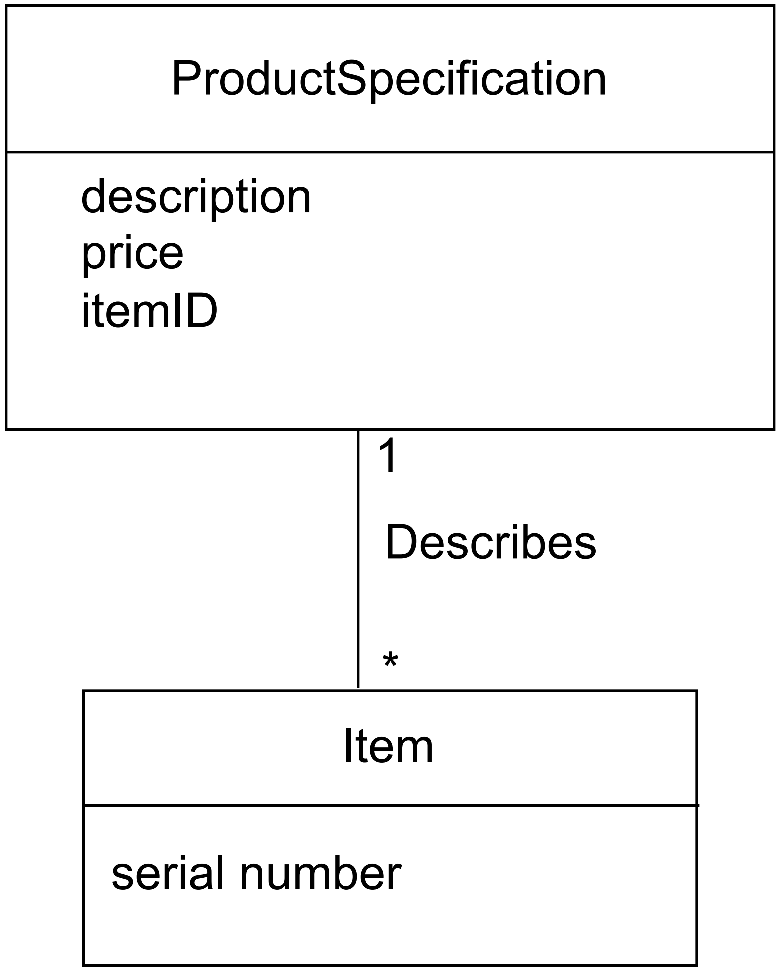
What’s wrong with this picture?

- Exercise:
  - So what is better?
  - Why?



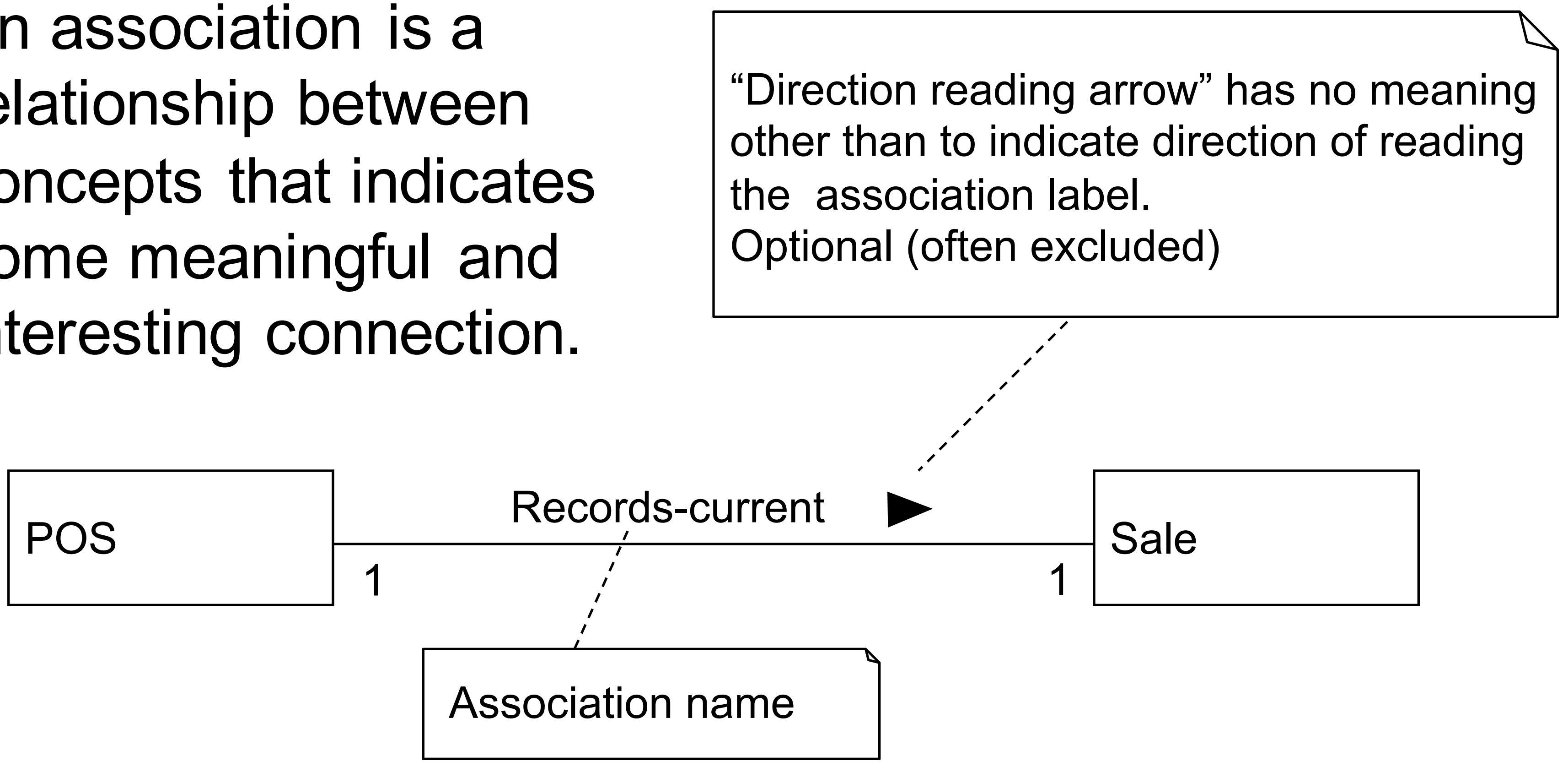
The Need for Specification or Description Conceptual Classes

- Add a specification or description concept when:
  - Deleting instances of things
    - 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.

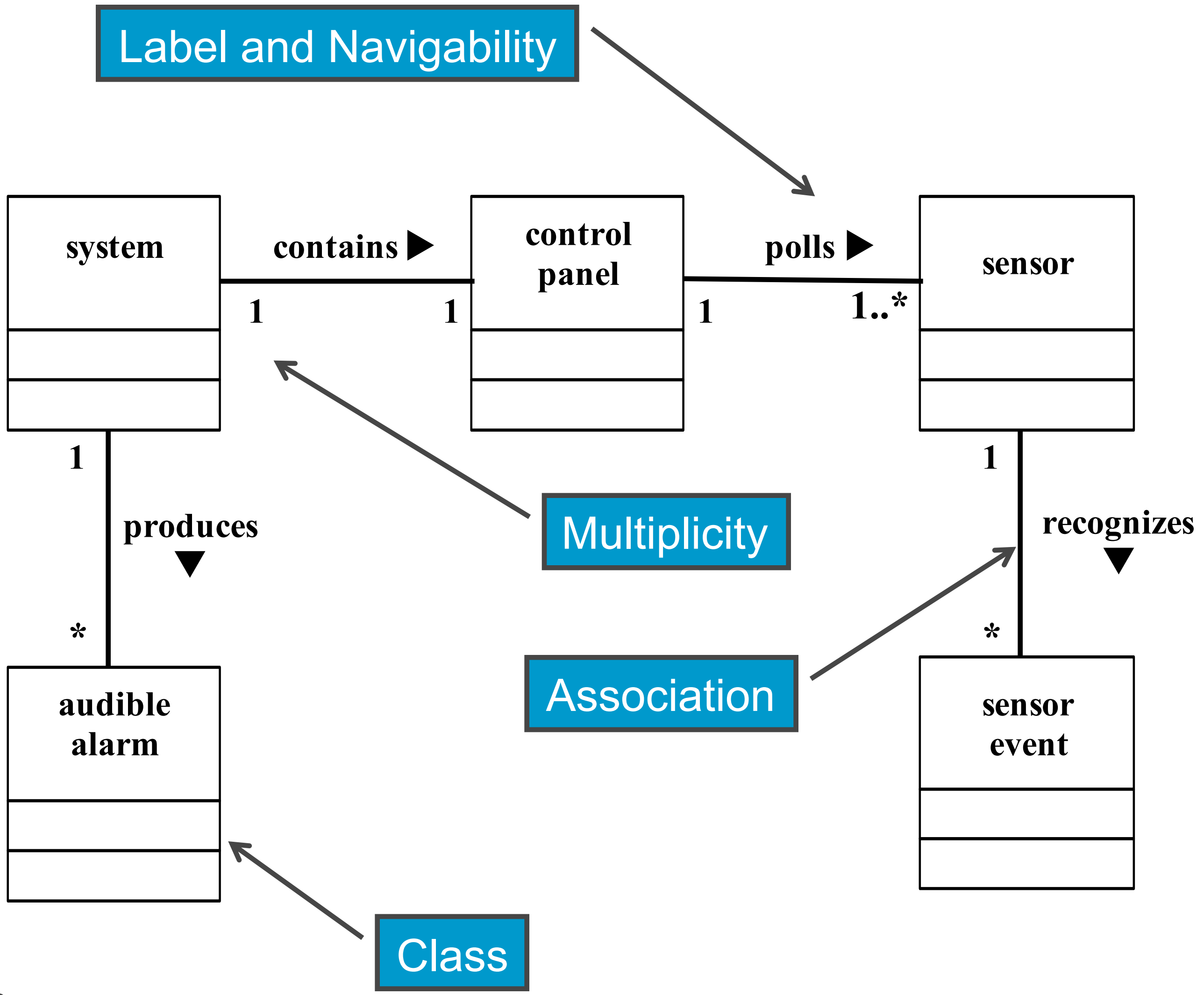


Adding Associations

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

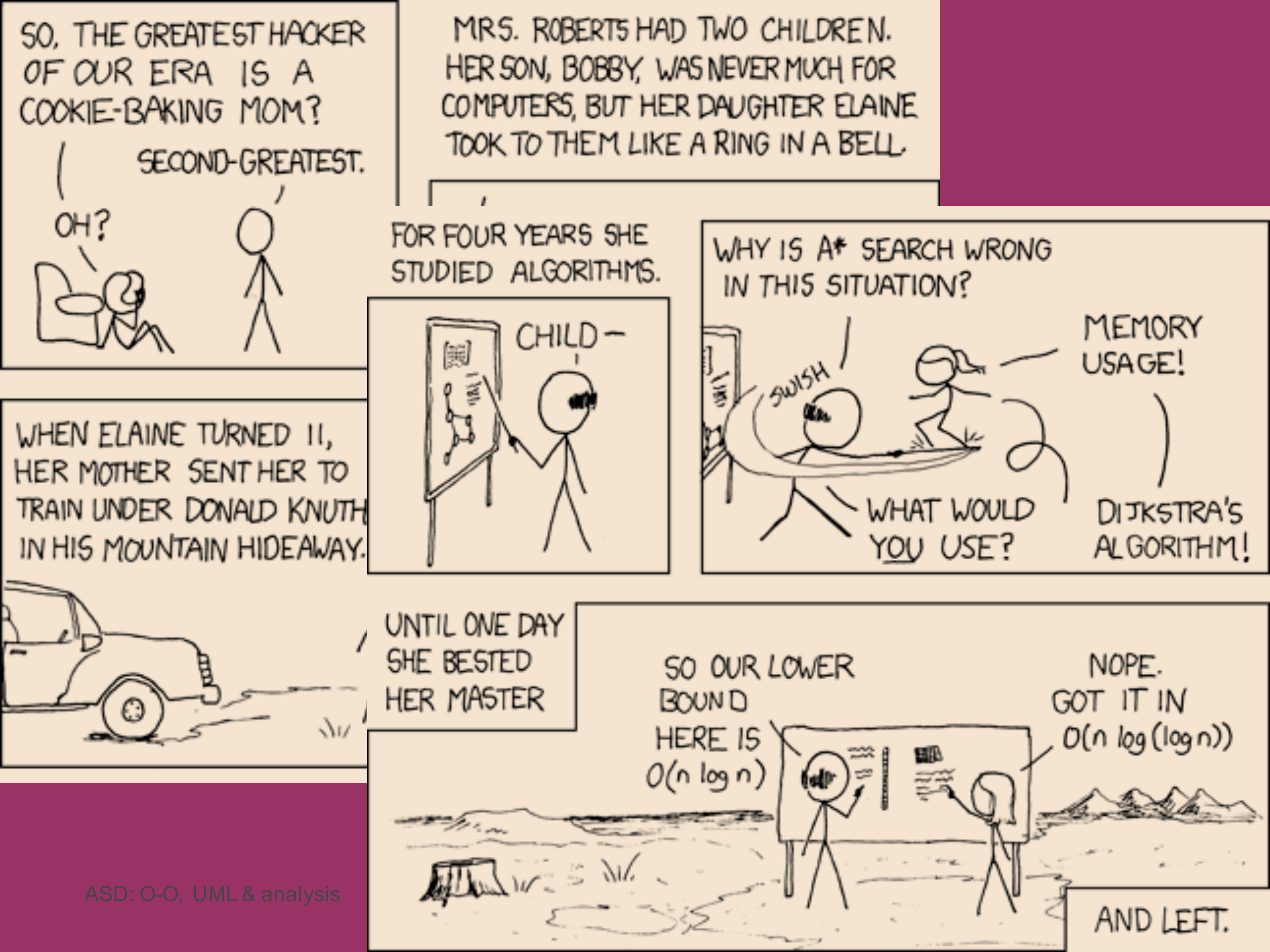

"Direction reading arrow" has no meaning other than to indicate direction of reading the association label.
Optional (often excluded)
POS
Records-current ▶
Sale
1
1
Association name

Example: Class Diagram



TO BE CONTINUED

Now onto the real thing …


SO, THE GREATEST HACKER OF OUR ERA IS A COOKIE-BAKING MOM?
SECOND-GREATEST.
OH?
MRS. ROBERTS HAD TWO CHILDREN. HER SON, BOBBY, WAS NEVER MUCH FOR COMPUTERS, BUT HER DAUGHTER ELAINE TOOK TO THEM LIKE A RING IN A BELL.
WHEN ELAINE TURNED 11, HER MOTHER SENT HER TO TRAIN UNDER DONALD KNUTH IN HIS MOUNTAIN HIDEAWAY.
FOR FOUR YEARS SHE STUDIED ALGORITHMS.
CHILD—
WHY IS A* SEARCH WRONG IN THIS SITUATION?
SWISH
MEMORY USAGE!
WHAT WOULD YOU USE?
DIJKSTRA'S ALGORITHM!
UNTIL ONE DAY SHE BESTED HER MASTER
SO OUR LOWER BOUND HERE IS O(n log n)
NOPE. GOT IT IN O(n log(log n))
AND LEFT.