# A Brief Introduction to UML

CS356 Object-Oriented Design and Programming
http://cs356.yusun.io
October 13, 2014

Yu Sun, Ph.D.
http://yusun.io
yusun@csupomona.edu

CAL POLY POMONA

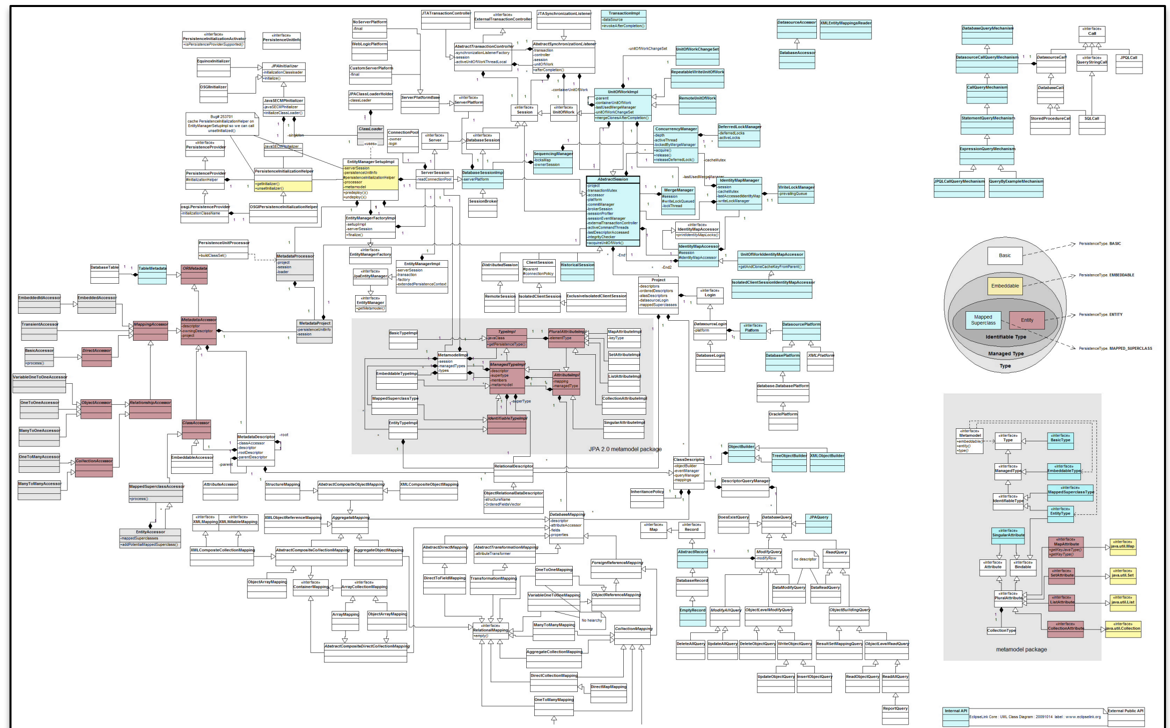
# UML in One Sentence

- The UML is a graphical language for
  - visualizing
  - specifying
  - constructing
  - documenting

  artifacts of a software-intensive system

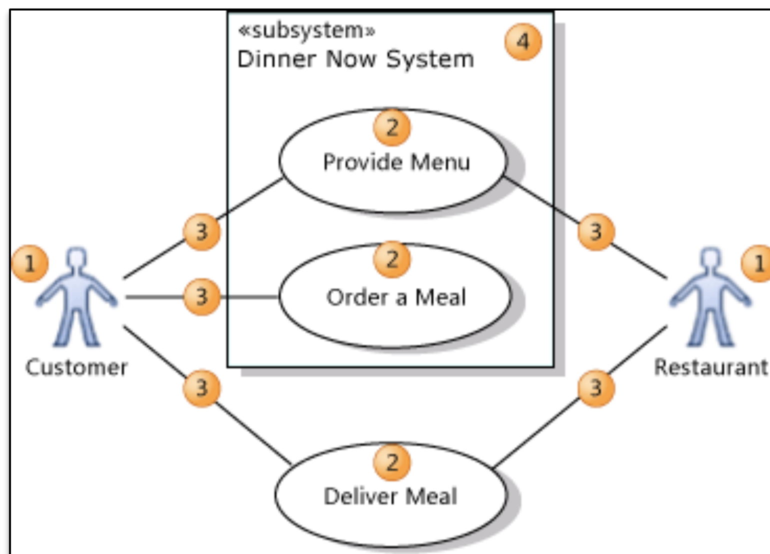
# Visualizing

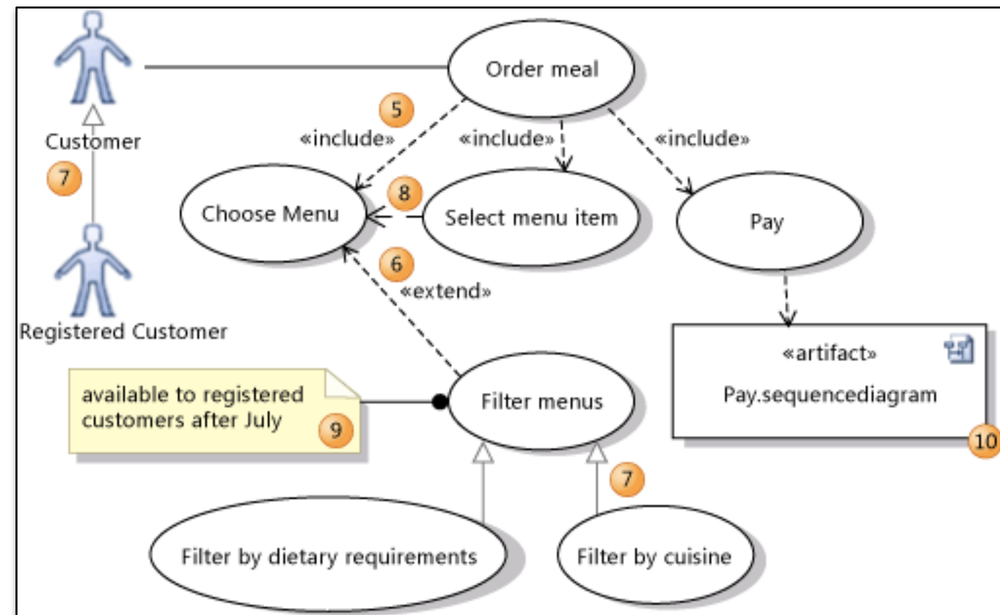
- Explicit model facilitates communication
- Some structures transcend what can be represented in programming language
- Each symbol has well-defined semantics behind it



# Specifying

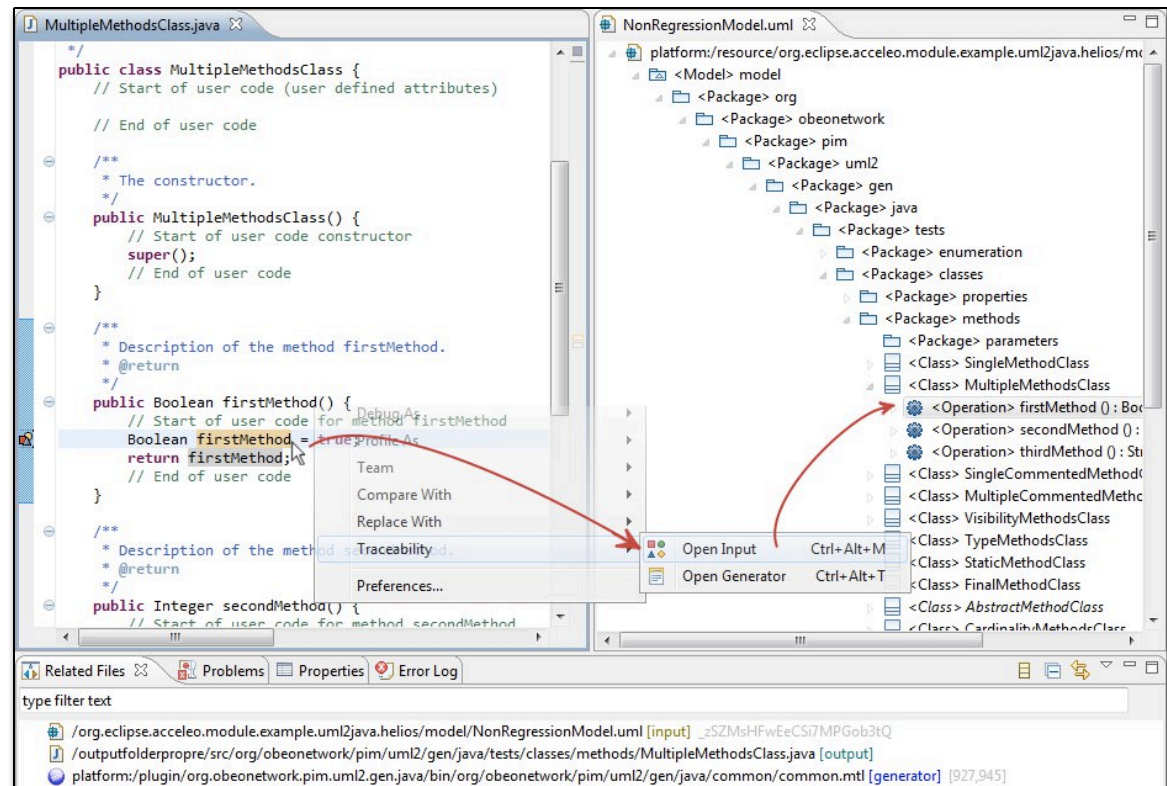
- The UML addresses the specification of all important analysis, design, and implementation decisions.



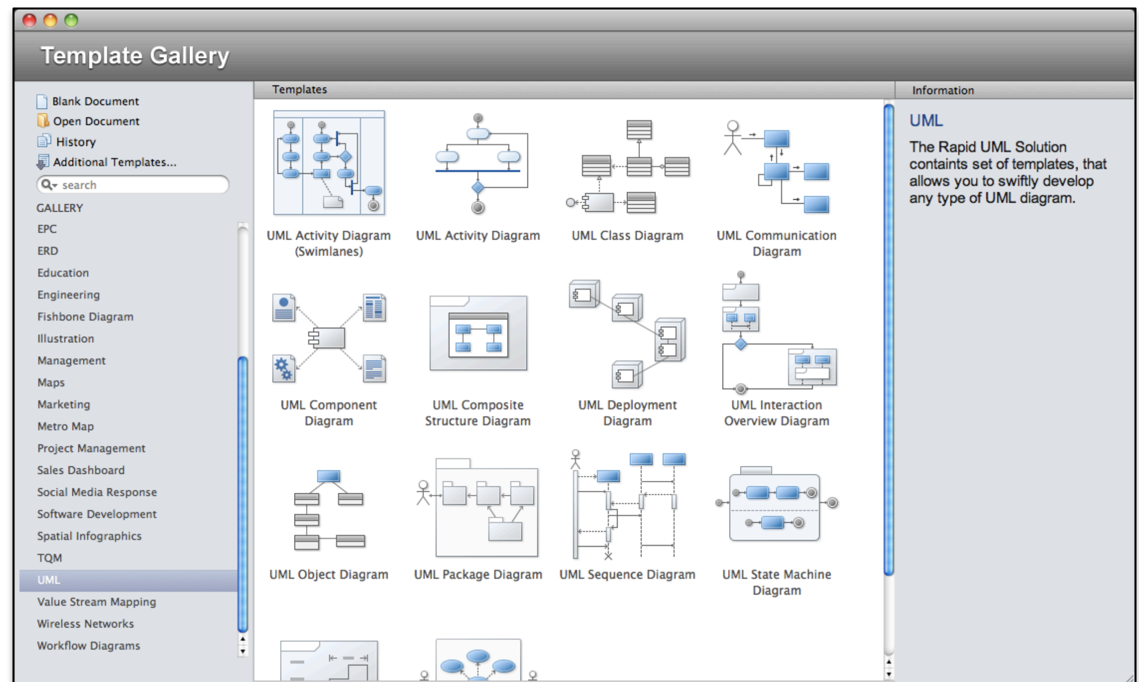
# Constructing

- Forward engineering: generation of code from model into programming language
- Reverse engineering: reconstructing model from implementation



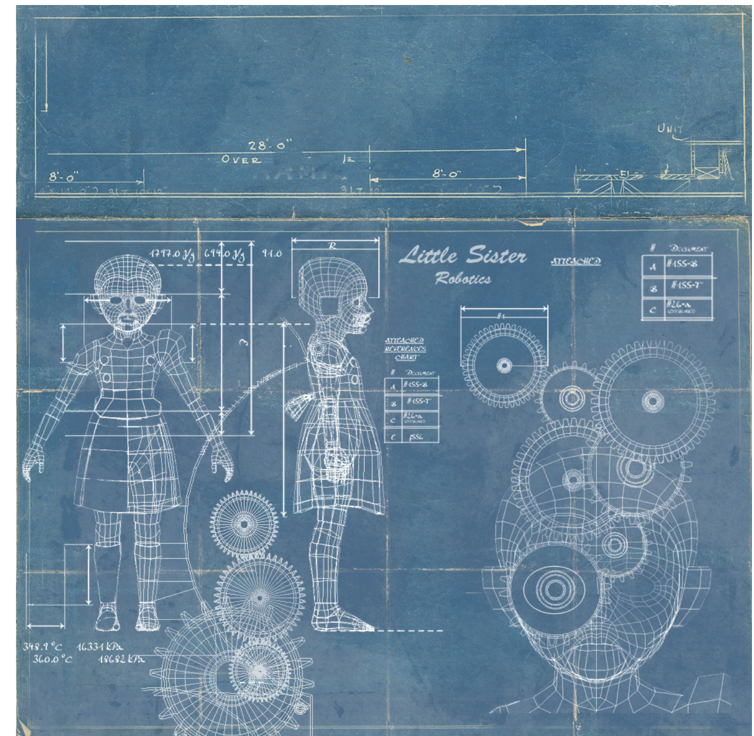
# Documenting

- Deliverables, such as requirements documents, functional specifications, and test plans
- Materials that are critical in controlling, measuring, and communicating about a system during development and after deployment

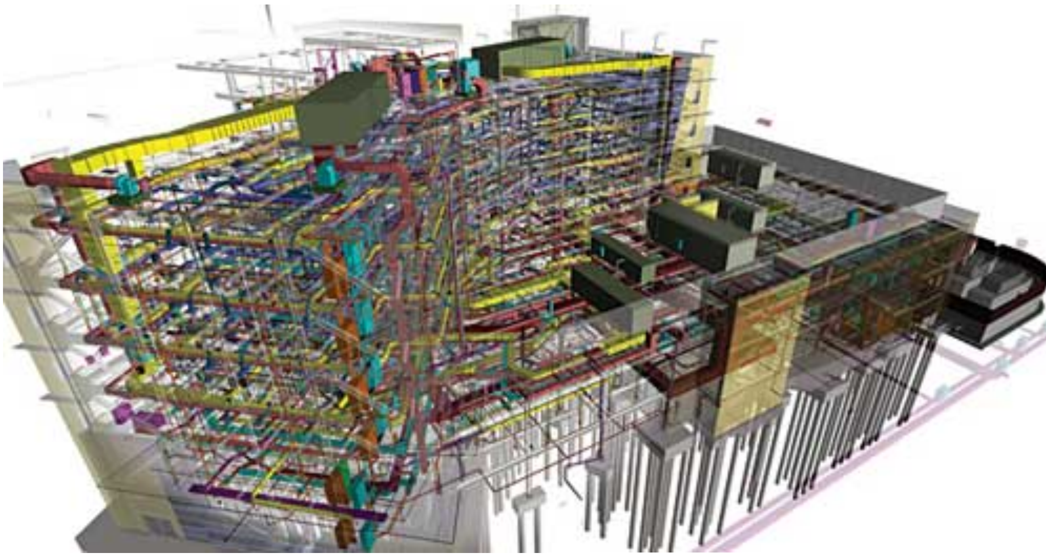


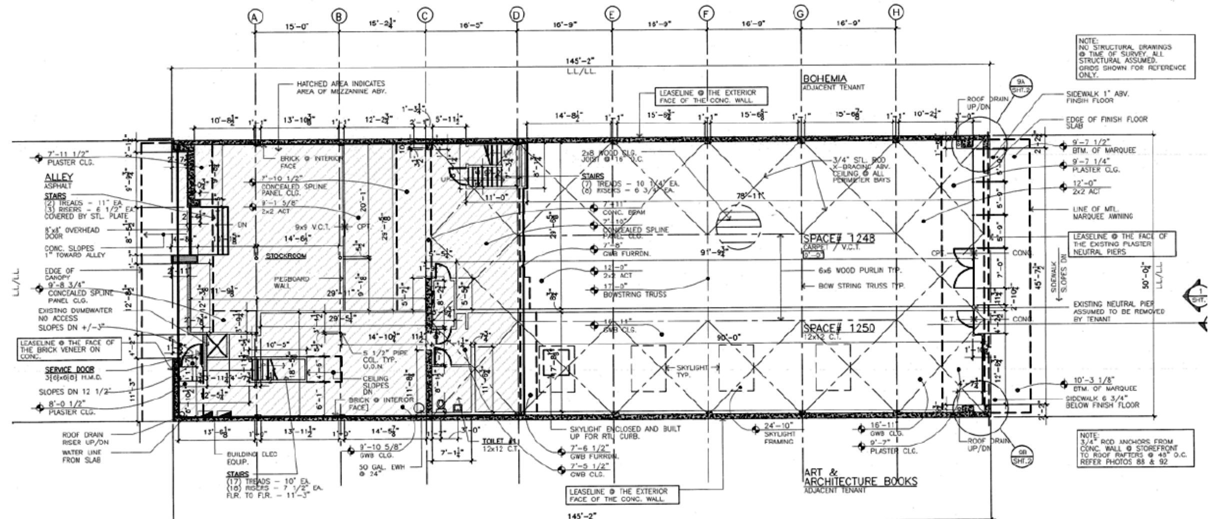
# UML and Blueprints

- UML provides a standard way to write a system's "blueprints" to account for
  - Conceptual things (business processes, system functions)
  - Concrete things (C++/Java classes, database schemas, reusable software components)



# Construction - Blueprint

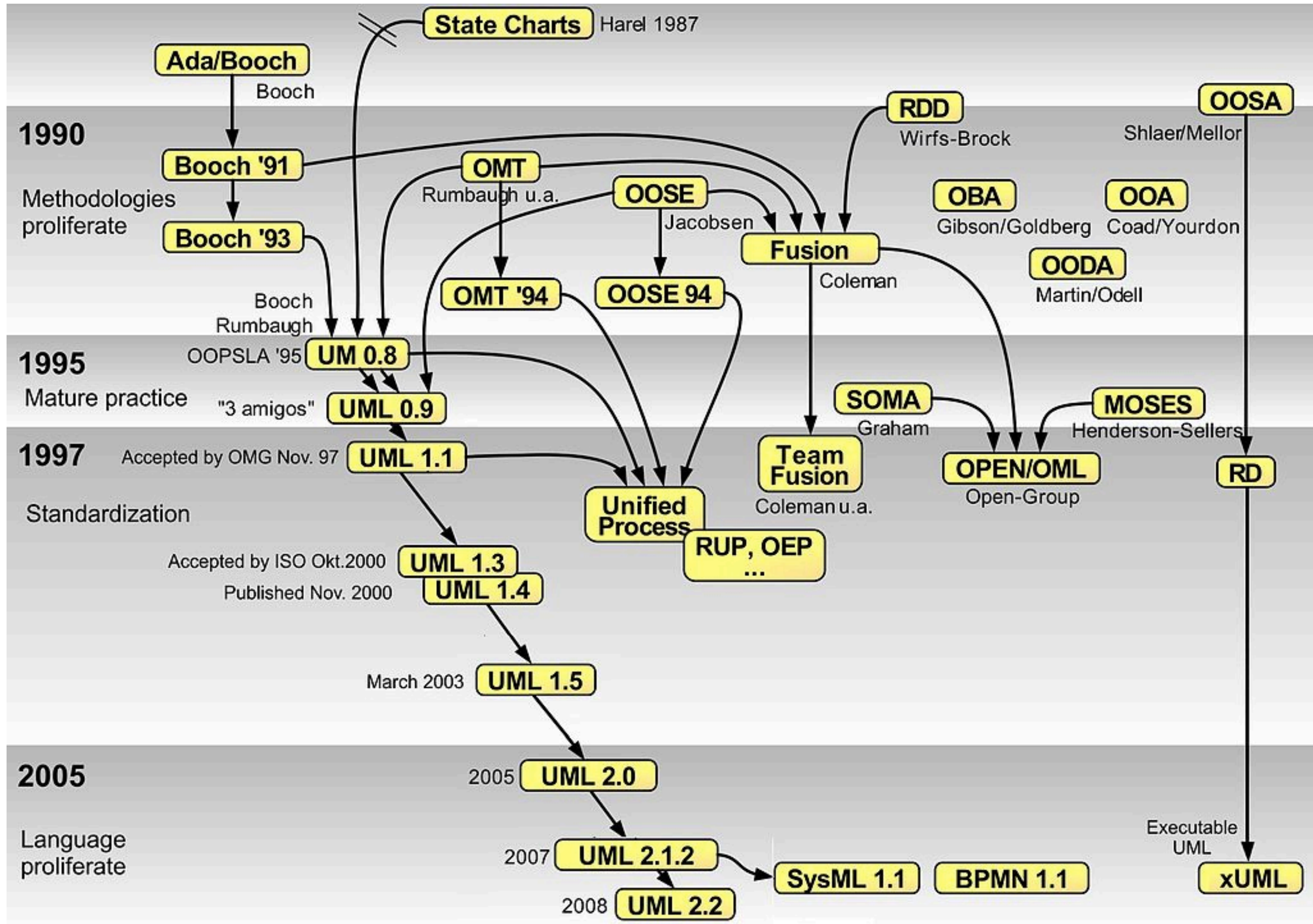




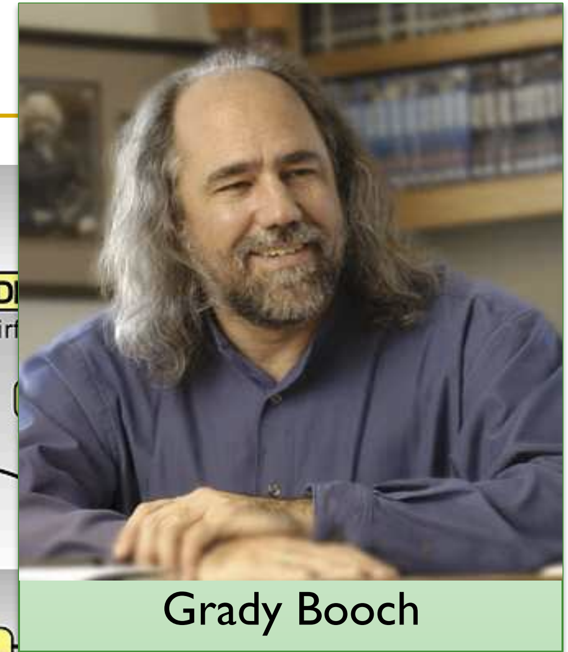
# Reasons to Model Software

- To communicate the desired structure and behavior of the system
- To visualize and control the system architecture
- To better understand the system and expose opportunities for simplification and reuse
- To manage risks

# UML History



# UML History


Grady Booch

State Charts
Harel 1987
Ada/Booch
Booch
RD
Wirf
1990
Booch '91
OMT
Methodologies proliferate
Rumbaugh u.a.
OOSE
Jacobsen
Booch '93
Fusion
OMT '94
OOSE 94
Coleman
Booch
Rumbaugh
1995
OOPSLA '95
UM 0.8
Mature practice
"3 amigos"
UML 0.9
SOMA
Graham
MOSES
Henderson-Sellers
1997
Accepted by OMG Nov. 97
UML 1.1
Team Fusion
Coleman u.a.
OPEN/OML
Open-Group
RD
Standardization
Unified Process
RUP, OEP ...
Accepted by ISO Okt.2000
UML 1.3
Published Nov. 2000
UML 1.4
March 2003
UML 1.5
2005
2005
UML 2.0
Language proliferate
Executable UML
2007
UML 2.1.2
SysML 1.1
BPMN 1.1
xUML
2008
UML 2.2

# Overview of UML Diagrams

## Structural

: element of spec. irrespective of time

- Class
- Component
- Deployment
- Object
- Composite structure
- Package

## Behavioral

: behavioral features of a system / business process

- Activity
- State machine
- Use case
- Interaction

## Interaction

: emphasize object interaction

- Communication
- Sequence
- Interaction overview
- Timing

# Overview of UML Diagrams

## Structural

: element of spec. irrespective of time

- **Class**
- Component
- Deployment
- Object
- Composite structure
- Package

## Behavioral

: behavioral features of a system / business process

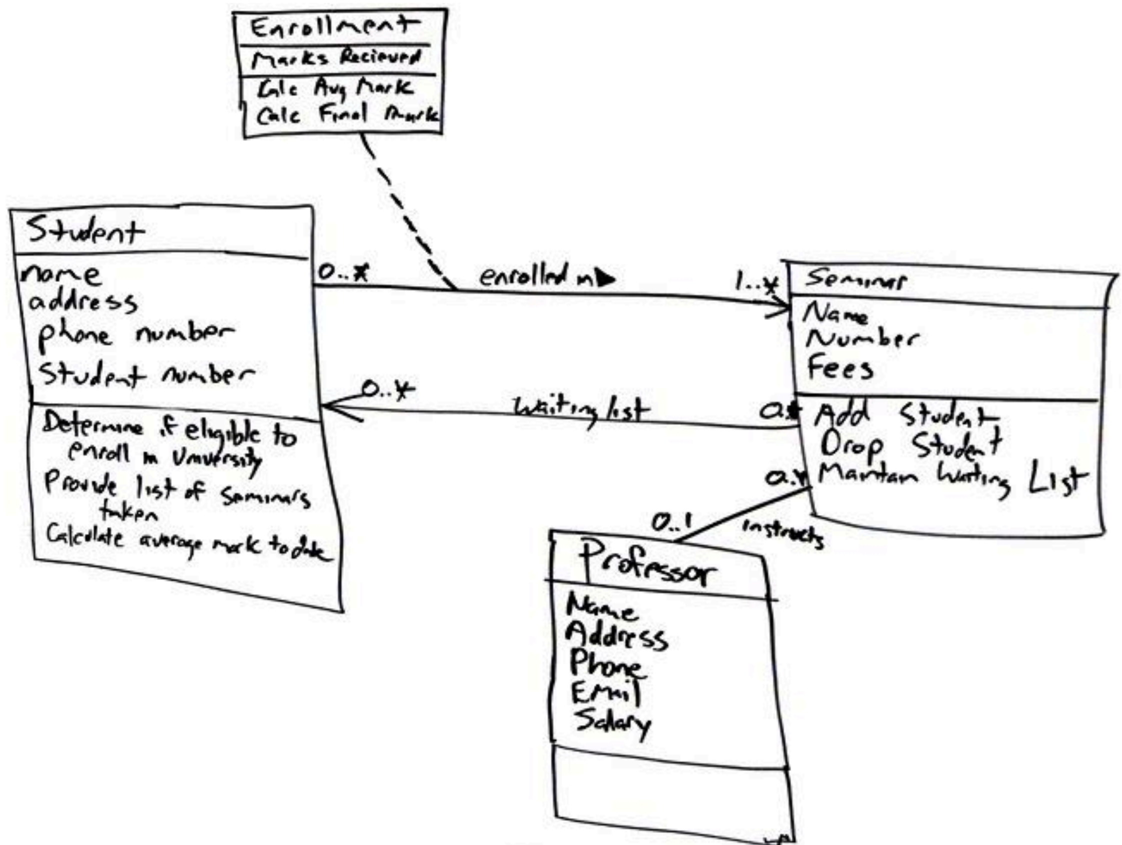
- Activity
- State machine
- **Use case**
- Interaction

## Interaction

: emphasize object interaction

- Communication
- **Sequence**
- Interaction overview
- Timing

# 1. Class Disgram



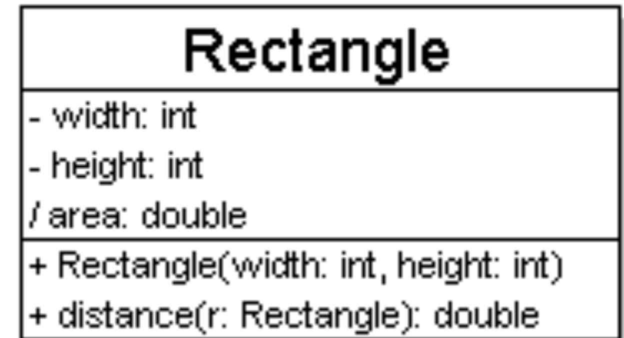
# UML Class Diagrams

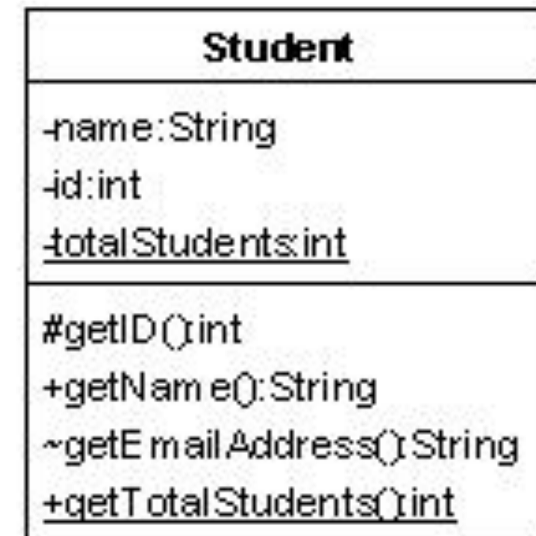
- What is a UML class diagram?
  - A picture of the classes in an OO system, their fields and methods, and connections between the classes that interact or inherit from each other

- What are some things that are not represented in a UML class diagram?
  - details of how the classes interact with each other
  - algorithmic details; how a particular behavior is implemented

# Diagram of One Class

- Class name in top of box
  - write <<interface>> on top of interfaces' names
  - use *italics* for an *abstract class* name
- Attributes (optional)
  - should include all fields of the object
- Operations / Methods (optional)
  - may omit trivial (get/set) methods
    - but don't omit any methods from an interface!
  - should not include inherited methods



# Class Attributes

- Attributes (fields, instance variables)
  - *visibility name* : *type* [*count*] = *default_value*
  - visibility:
    - + public
    - # protected
    - - private
    - ~ package (default)
    - / derived
  - underline static attributes
  - **derived attribute**: not stored, but can be computed from other attribute values
  - attribute example:
    - balance : double = 0.00

**Rectangle**

- width: int
- height: int
/ area: double
+ Rectangle(width: int, height: int)
+ distance(r: Rectangle): double

**Student**

-name:String
-id:int
-totalStudents:int
#getID():int
+getName():String
~getEmailAddress():String
+getTotalStudents():int

# Class Operations / Methods

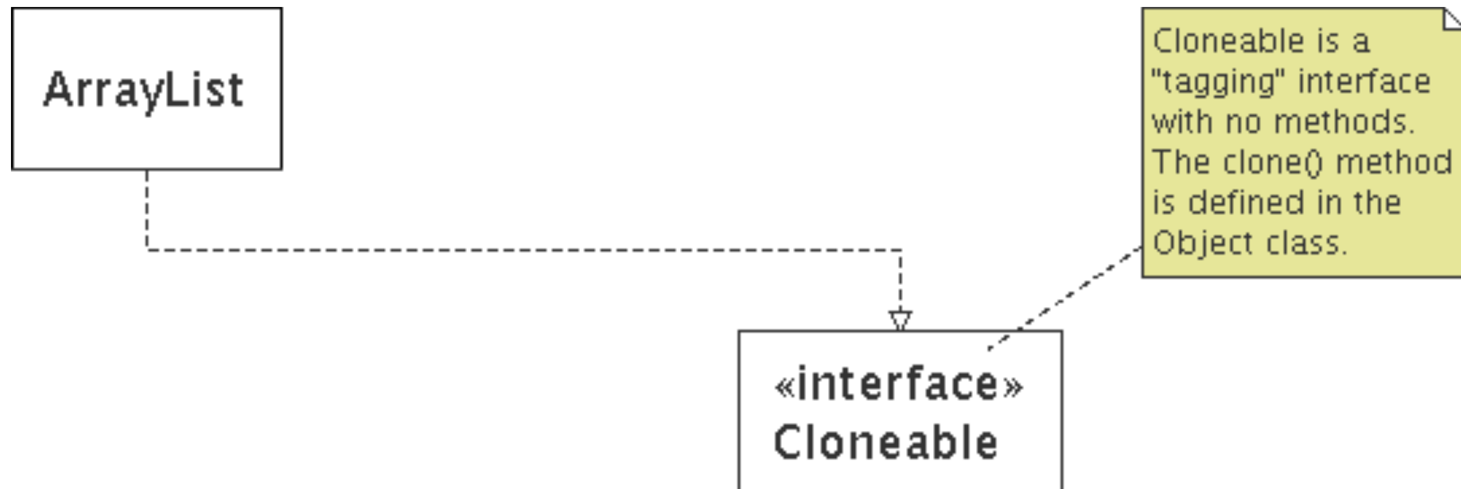
- Operations / Methods
  - *visibility name* (*parameters*) : *return_type*
  - visibility:
    - \+ public
    - \# protected
    - \- private
    - ~ package (default)
  - underline <u>static methods</u>
  - parameter types listed as (name: type)
  - omit *return_type* on constructors and when return type is void
  - method example:
    \+ distance(p1: Point, p2: Point): double

| Rectangle |
|---|
| - width: int<br>- height: int<br>/ area: double |
| + Rectangle(width: int, height: int)<br>+ distance(r: Rectangle): double |

| Student |
|---|
| -name:String<br>-id:int<br><u>-totalStudents:int</u> |
| #getID():int<br>+getName():String<br>~getEmailAddress():String<br><u>+getTotalStudents():int</u> |

# Comments

- Represented as a folded note, attached to the appropriate class/method by a dashed line

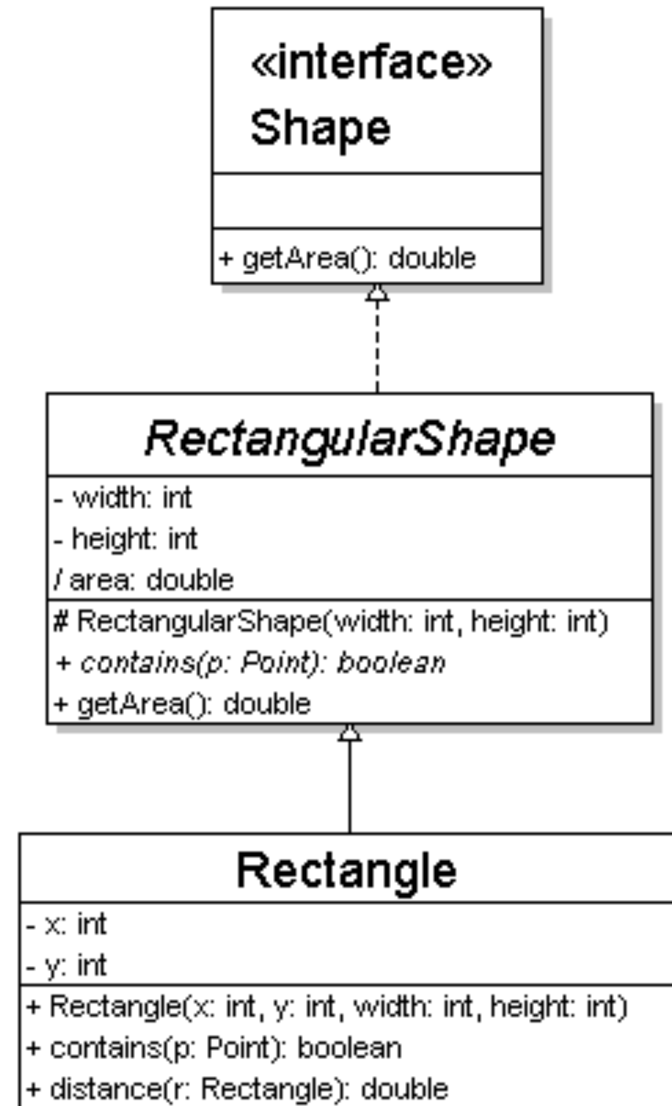


# Relationships in Class Diagram

- **Generalization**: an inheritance relationship
  - inheritance between classes
  - interface implementation
- **Association**: a usage relationship
  - dependency
  - aggregation
  - composition

# Generalization

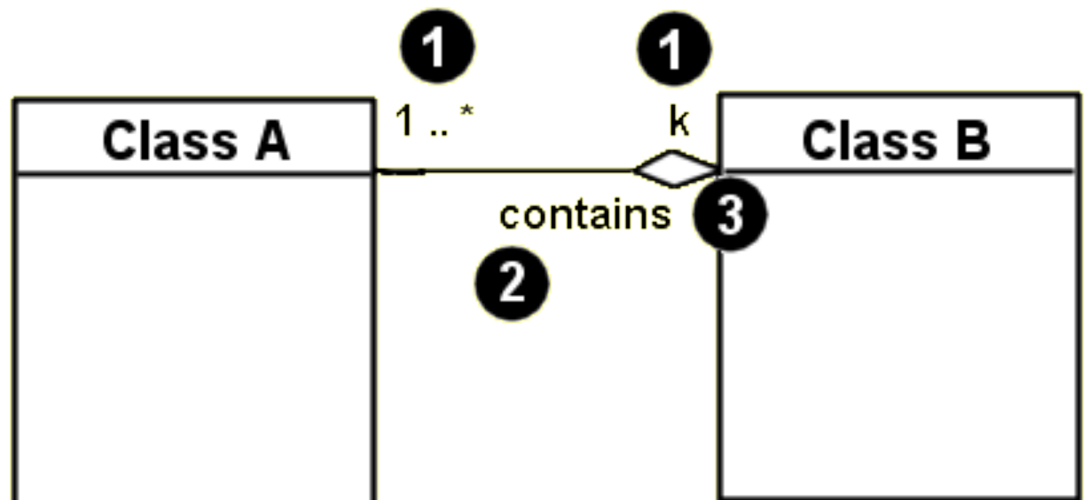
- Generalization (Inheritance)
  - hierarchies drawn top-down with arrows pointing upward to parent
  - line/arrow styles differ, based on whether parent is a(n):
    - class:
      solid line, black arrow
    - abstract class:
      solid line, white arrow
    - interface:
      dashed line, white arrow
  - we often don't draw trivial / obvious generalization relationships, such as drawing the Object class as a parent



# Association

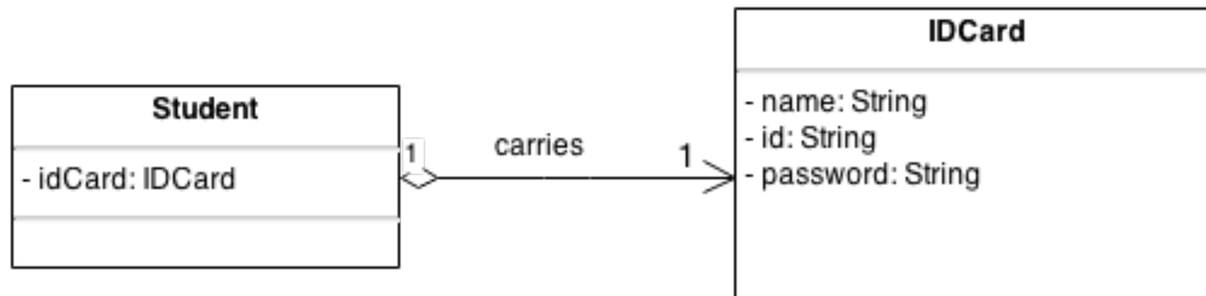
- Associational (usage) relationships

1. multiplicity (how many are used)
   - * ⇒ 0, 1, or more
   - 1 ⇒ 1 exactly
   - 2..4 ⇒ between 2 and 4, inclusive
   - 3..* ⇒ 3 or more
2. name (what relationship the objects have)
3. navigability (direction)

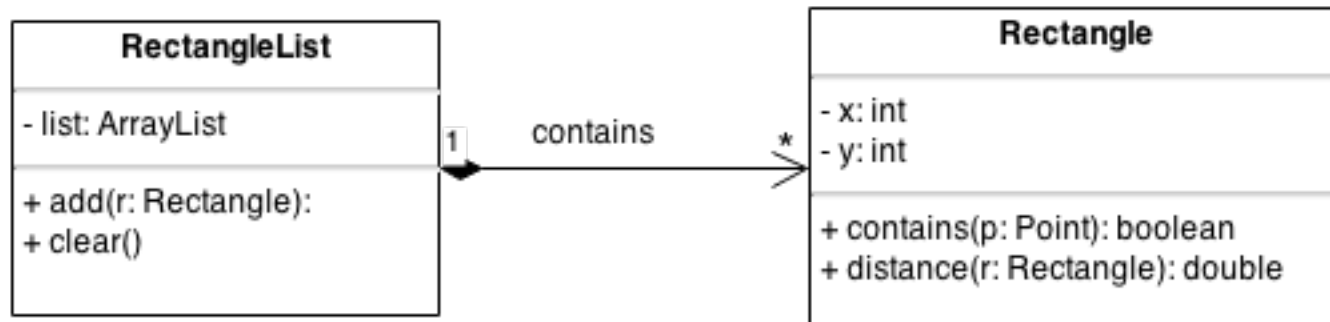


# Multiplicity of Associations

- one-to-one
  - each student must carry exactly one ID card

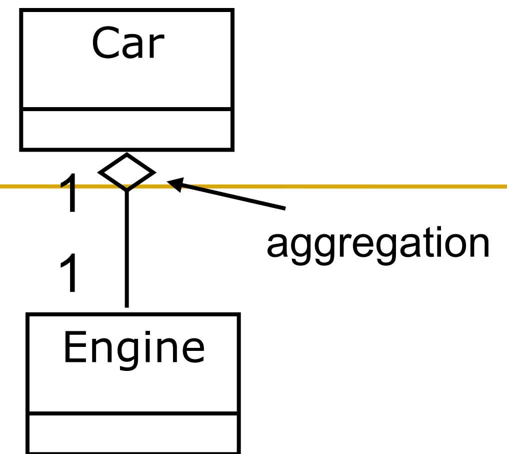



- one-to-many
  - one rectangle list can contain many rectangles

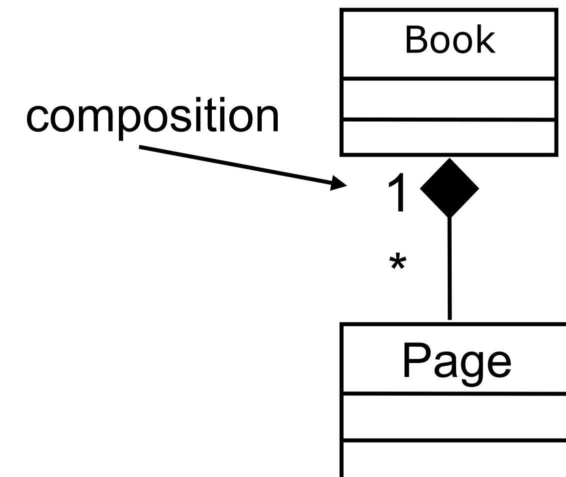


# Association Types

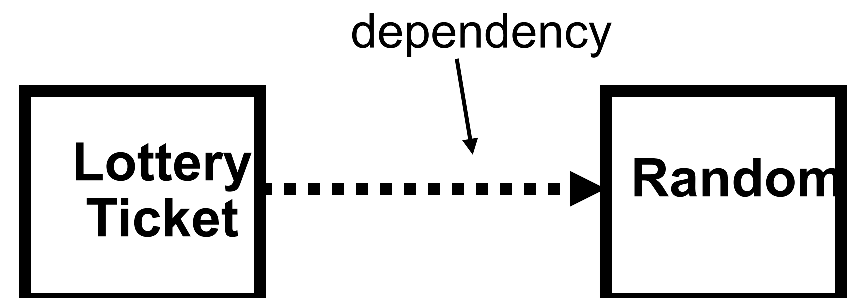
- **Aggregation**: ”has-a/is part of"
  - symbolized by a clear white diamond




- **Composition**: "is entirely made of"
  - stronger version of aggregation
  - the parts live and die with the whole
  - symbolized by a black diamond

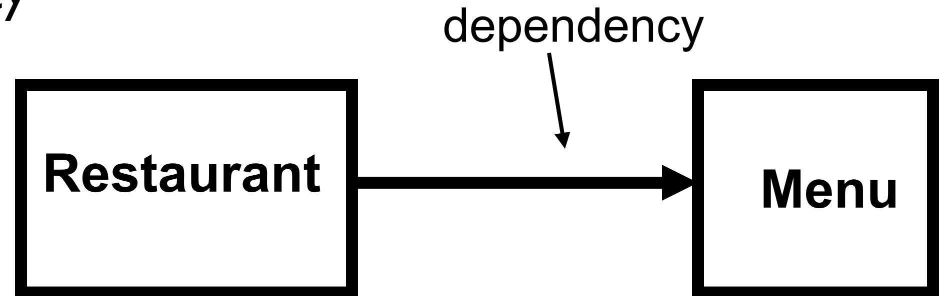


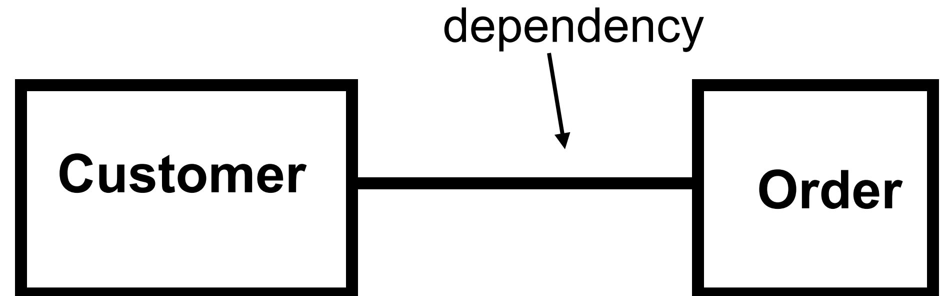

- **Dependency**: "uses temporarily"
  - symbolized by dotted line
  - often is an implementation detail, not an intrinsic part of that object's state



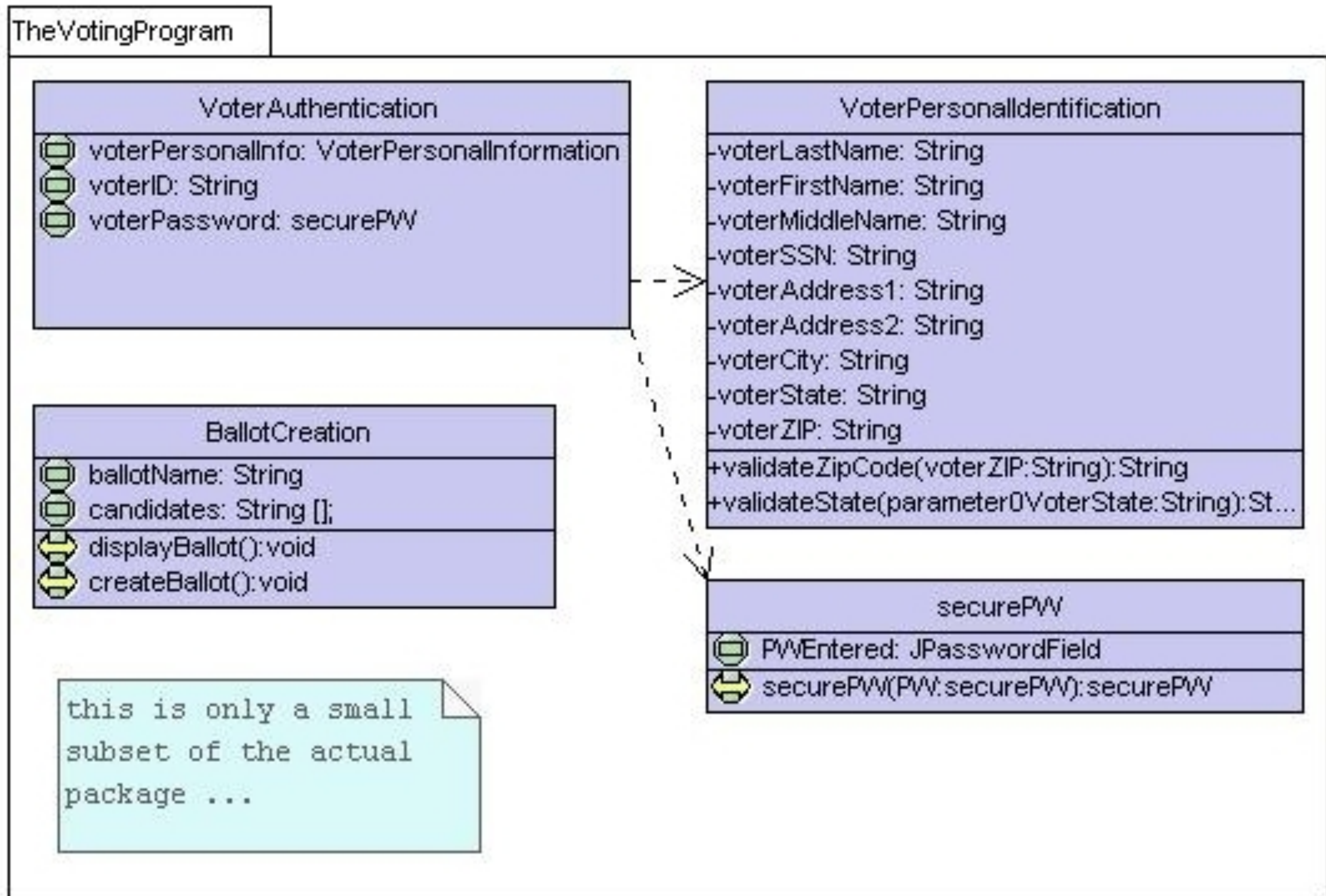
# Association Types

- **Simple association**:
  - generic format
  - some type of link or dependency

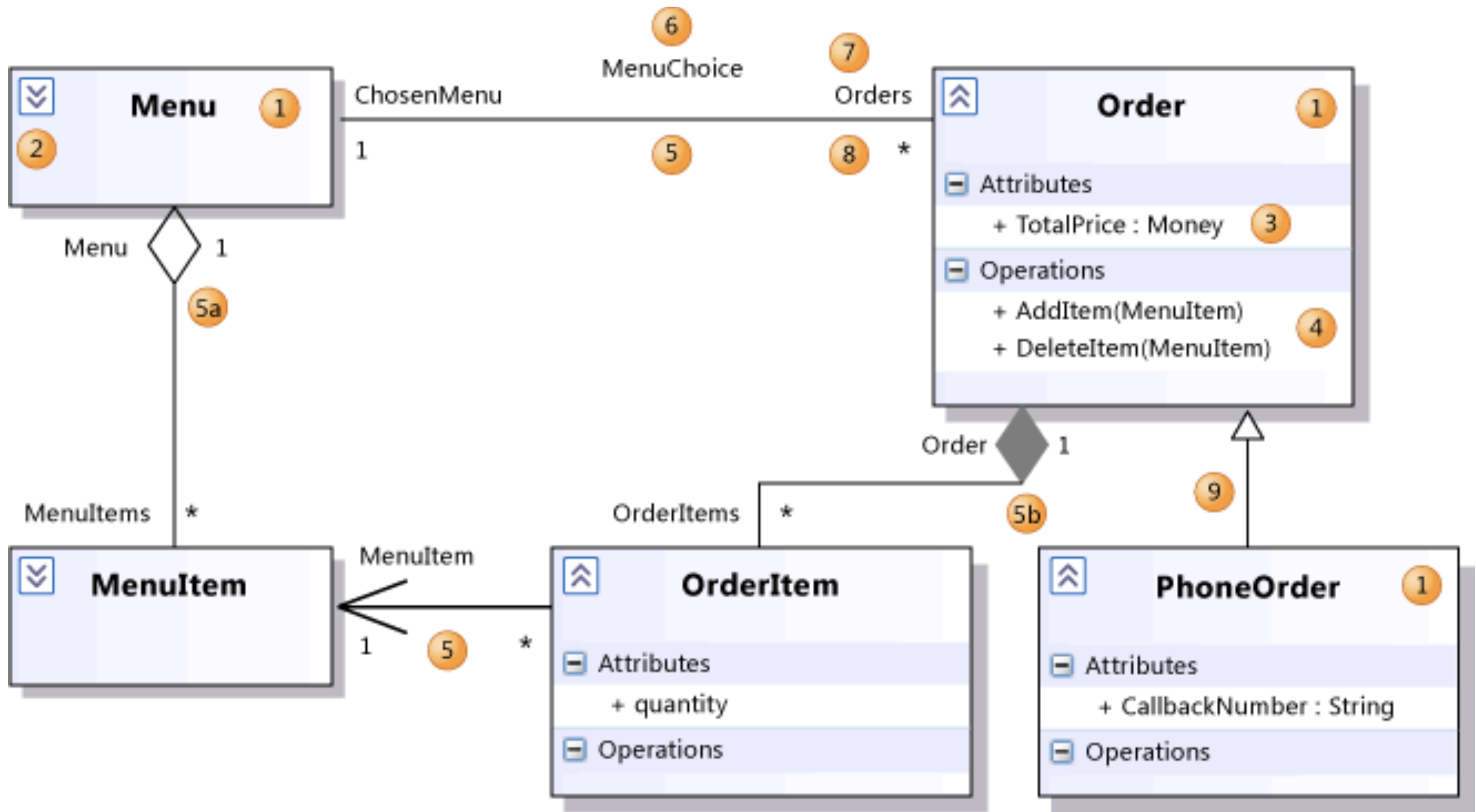


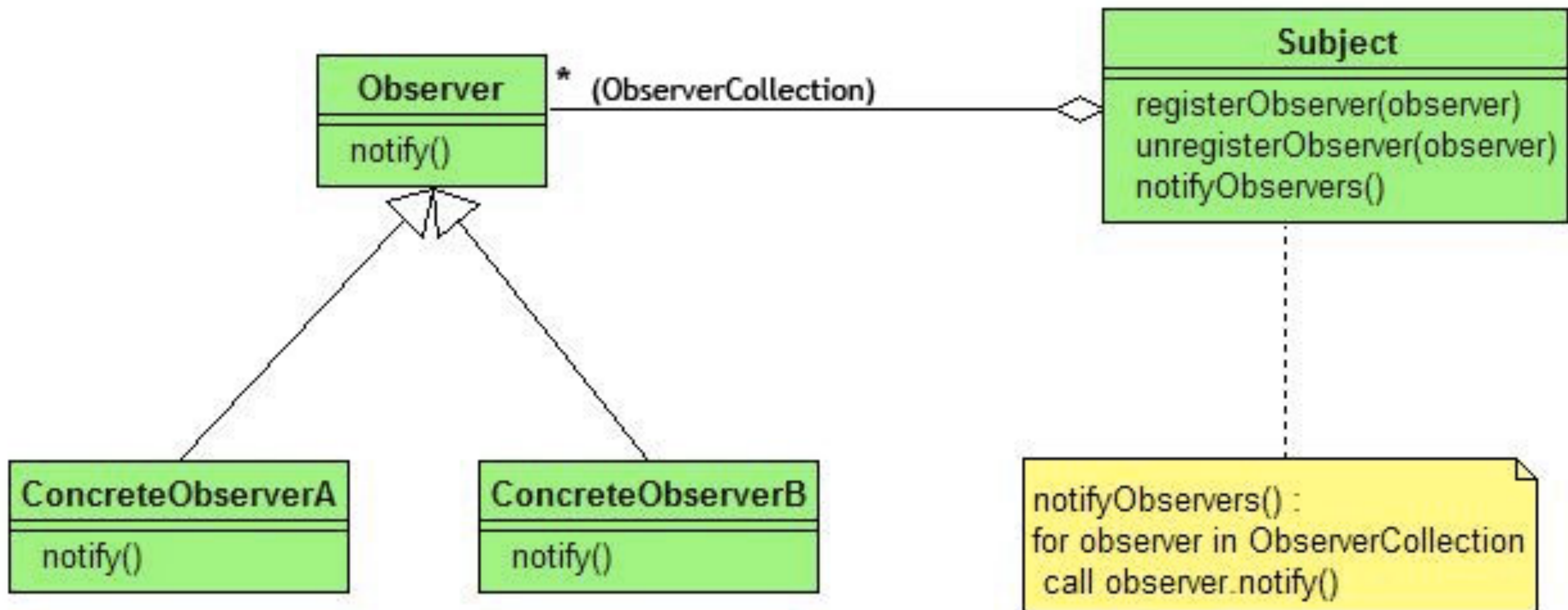
# Class Diagram Example 1



# Class Diagram Example 2



# Class Diagram Example 3



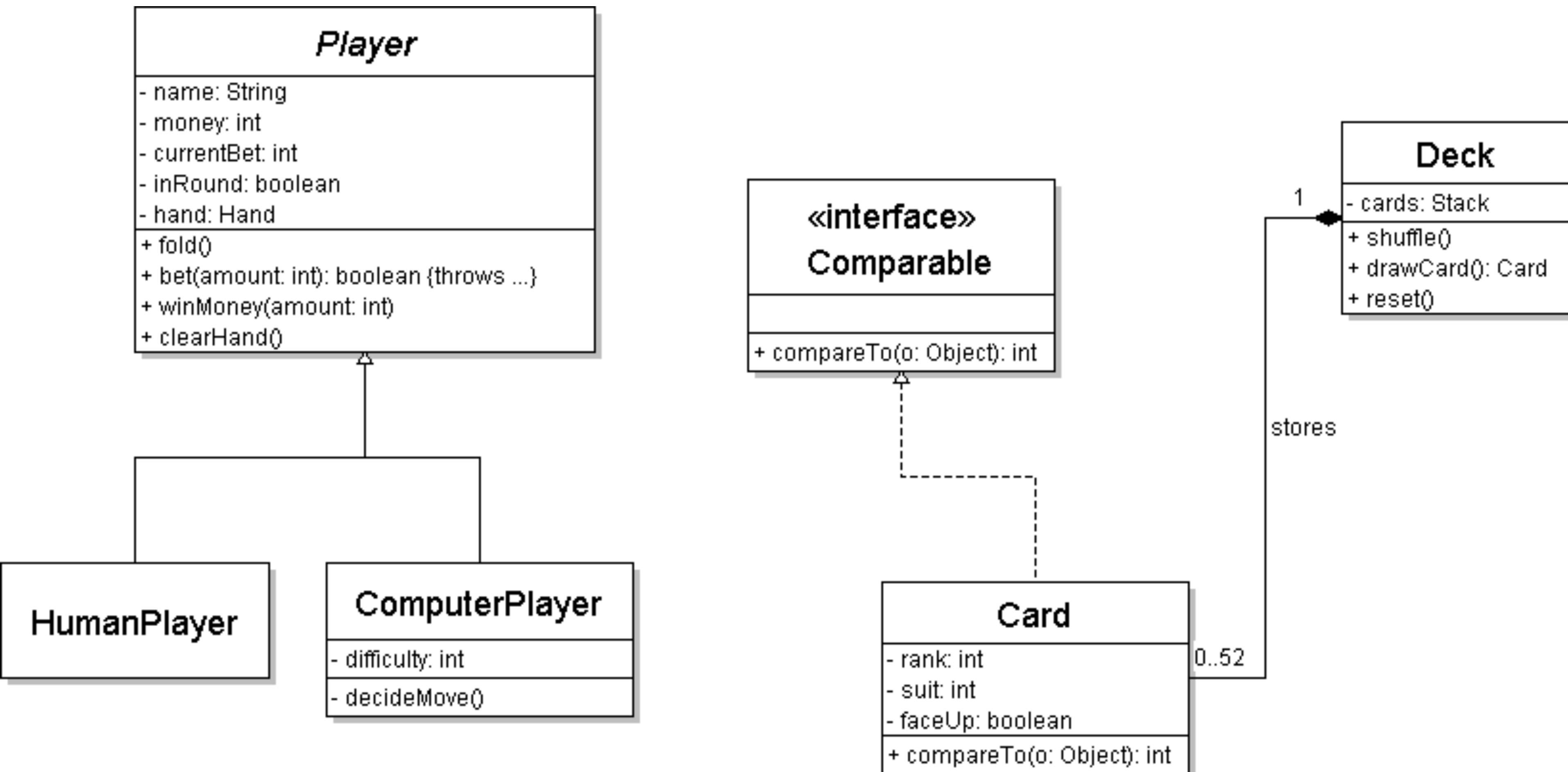
# Tools for Creating UML Diagrams

- Violet
  - http://horstmann.com/violet/
- Google Drive
  - http://drive.google.com
- Rational Rose
  - http://www.rational.com/

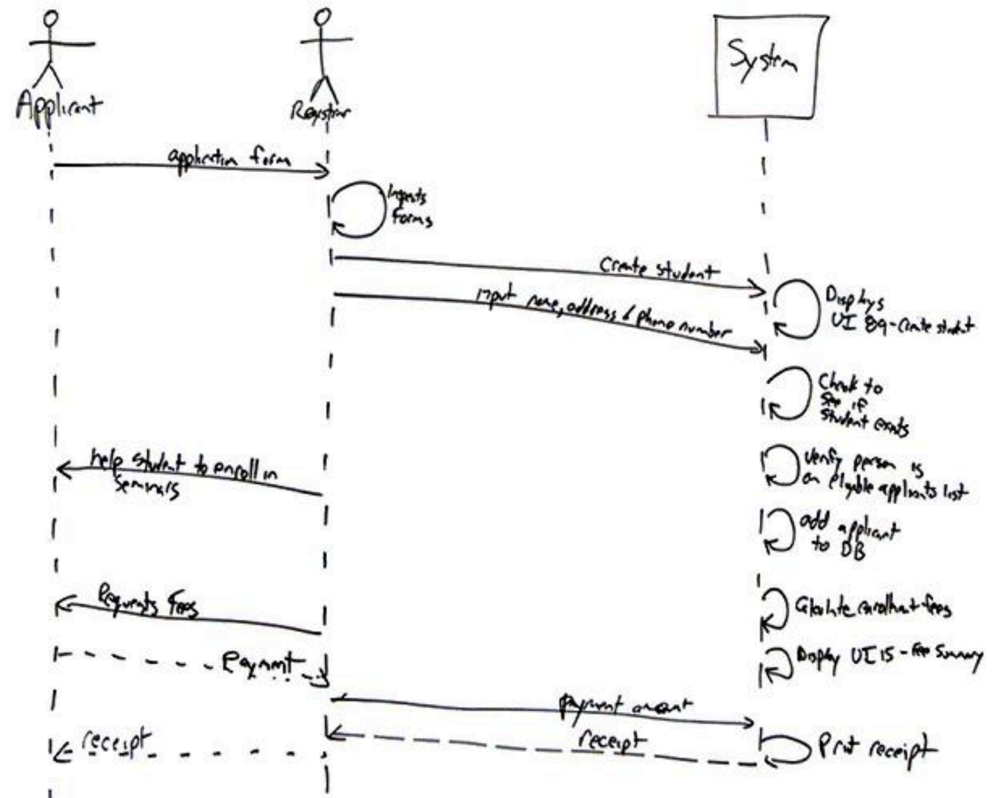
# Class Design Exercise

- Design Texas Hold 'em poker game system:
  - Human or computer players
  - Each player has a name and stack of chips
  - Computer players have a difficulty setting: easy, medium, hard
  - A deck contains 52 cards
  - For each hand:
    - Dealer collects ante from appropriate players, shuffles the deck, and deals each player a hand of 2 cards from the deck
    - A betting round occurs, followed by dealing 3 shared cards from the deck
    - As shared cards are dealt, more betting rounds occur, where each player can fold, check, or raise
    - At the end of a round, if more than one player is remaining, players' hands are compared, and the best hand wins the pot of all chips bet so far

# Class Design Exercise

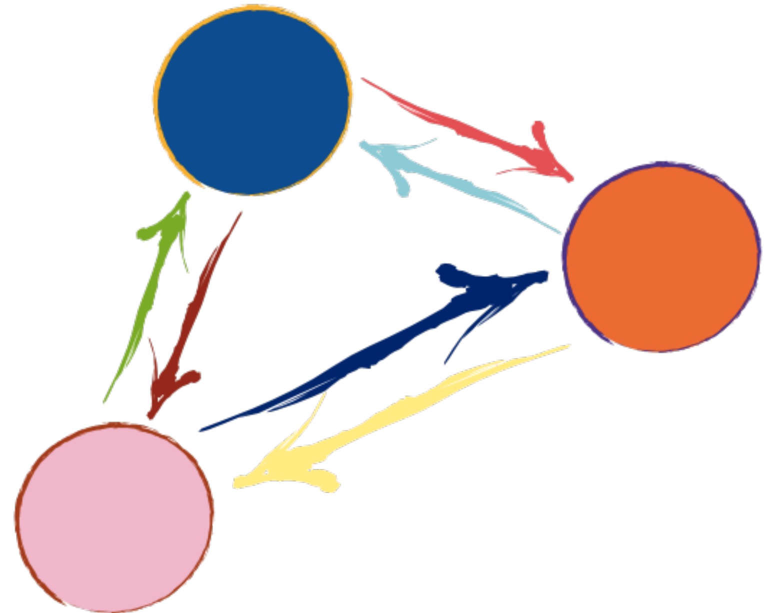


# 2. Sequence Diagram

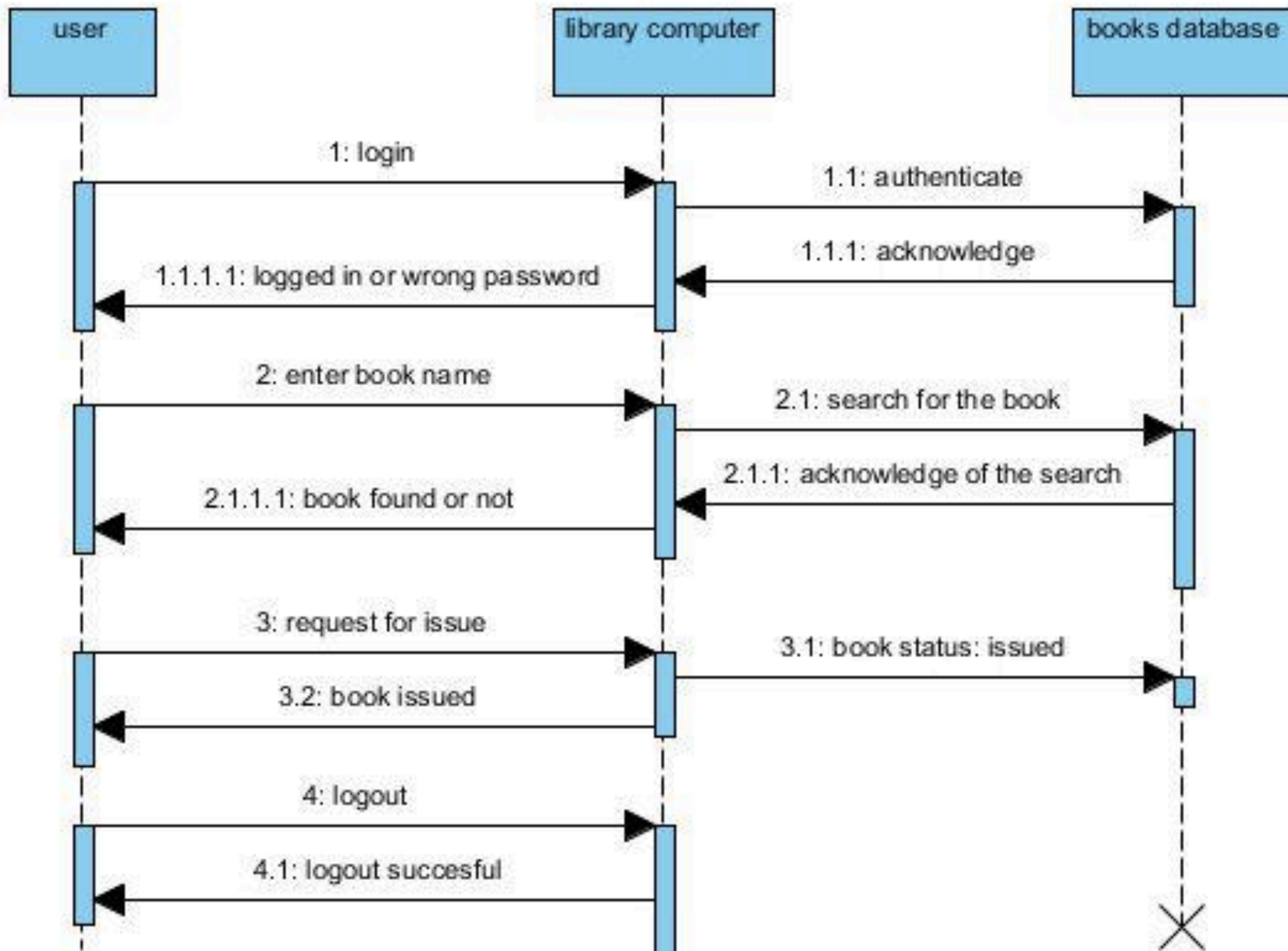


# UML sequence diagrams

- **Sequence diagram**: an "interaction diagram" that models a single scenario executing in the system
  - perhaps 2nd most used UML diagram (behind class diagram)



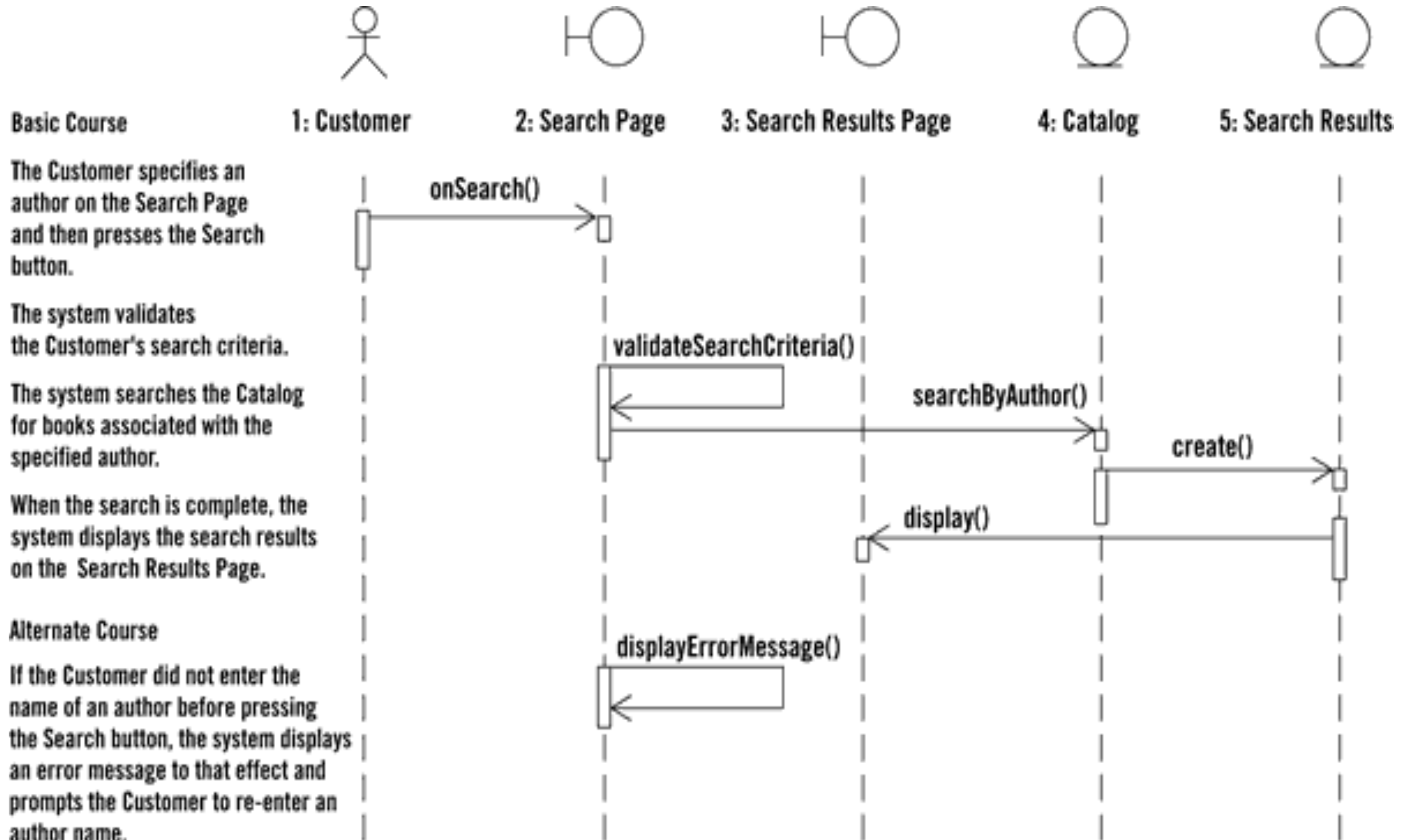
# Easy to Understand



# Key Parts in a Sequence Diagram

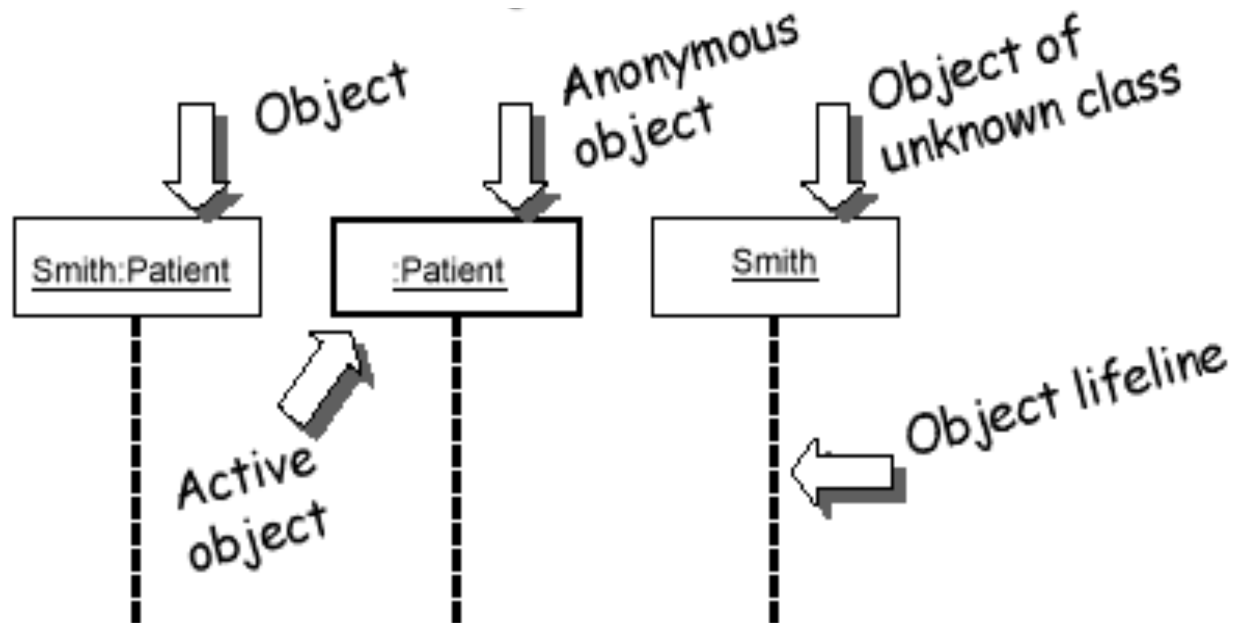
- **Participant**: an object or entity that acts in the sequence diagram
  - sequence diagram starts with an unattached "found message" arrow
- **Message**: communication between participant objects
- The axes in a sequence diagram:
  - horizontal: which object/participant is acting
  - vertical: time (down -> forward in time)

# Sequence Diagram from Use Case



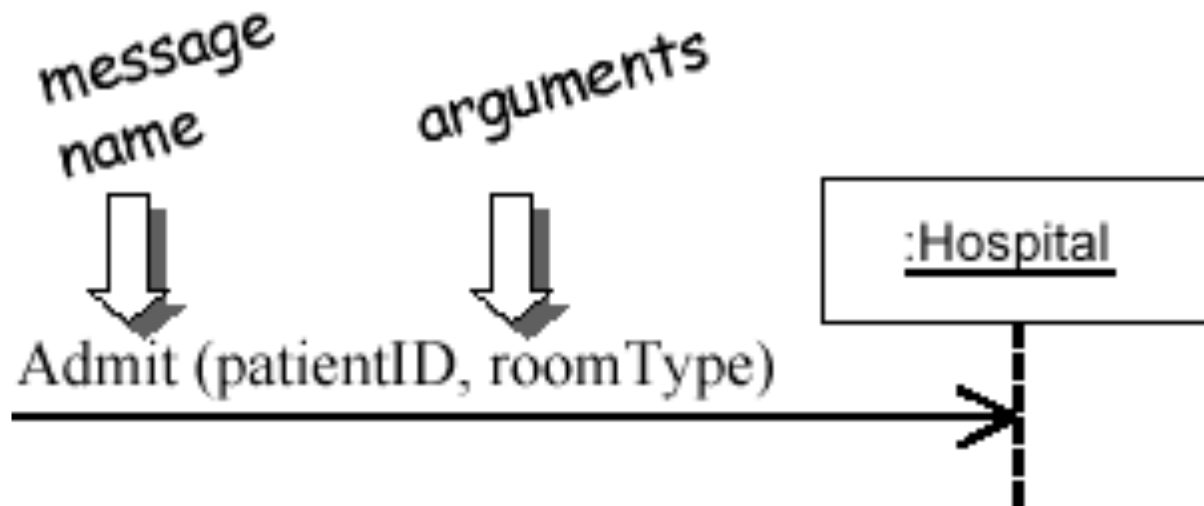
# Representing Objects

- Squares with object type, optionally preceded by object name and colon
  - write object's name if it clarifies the diagram
  - object's "life line" represented by dashed vert. line



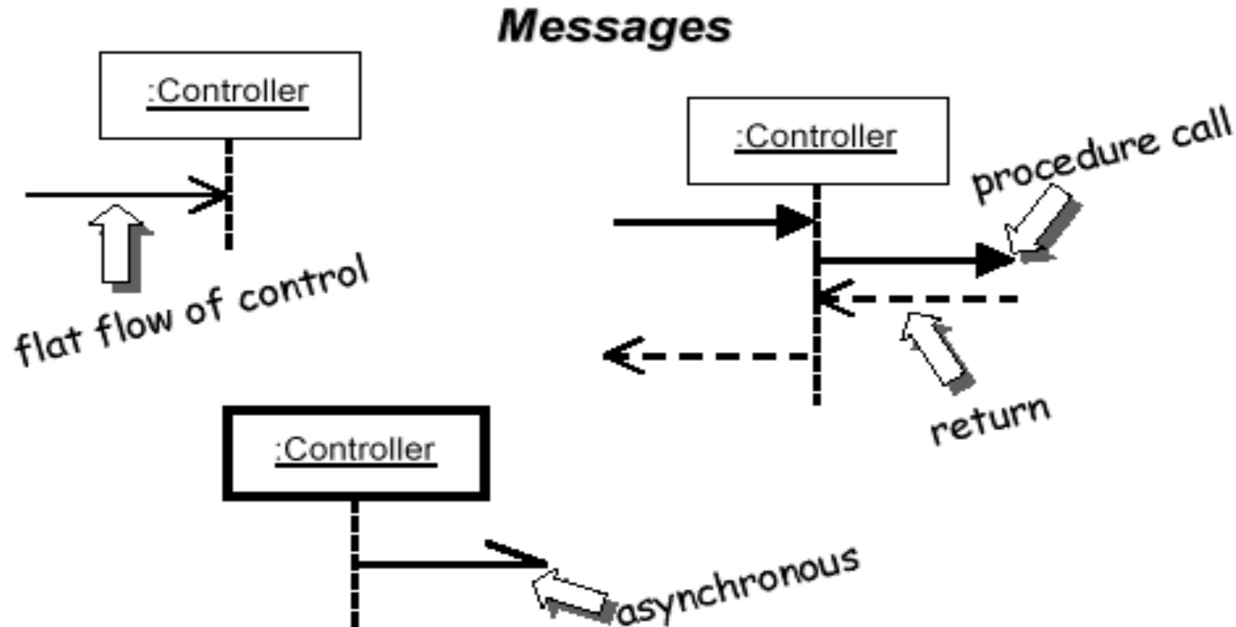
# Messages between Objects

- Message (method call) indicated by horizontal arrow to other object
  - write message name and arguments above arrow



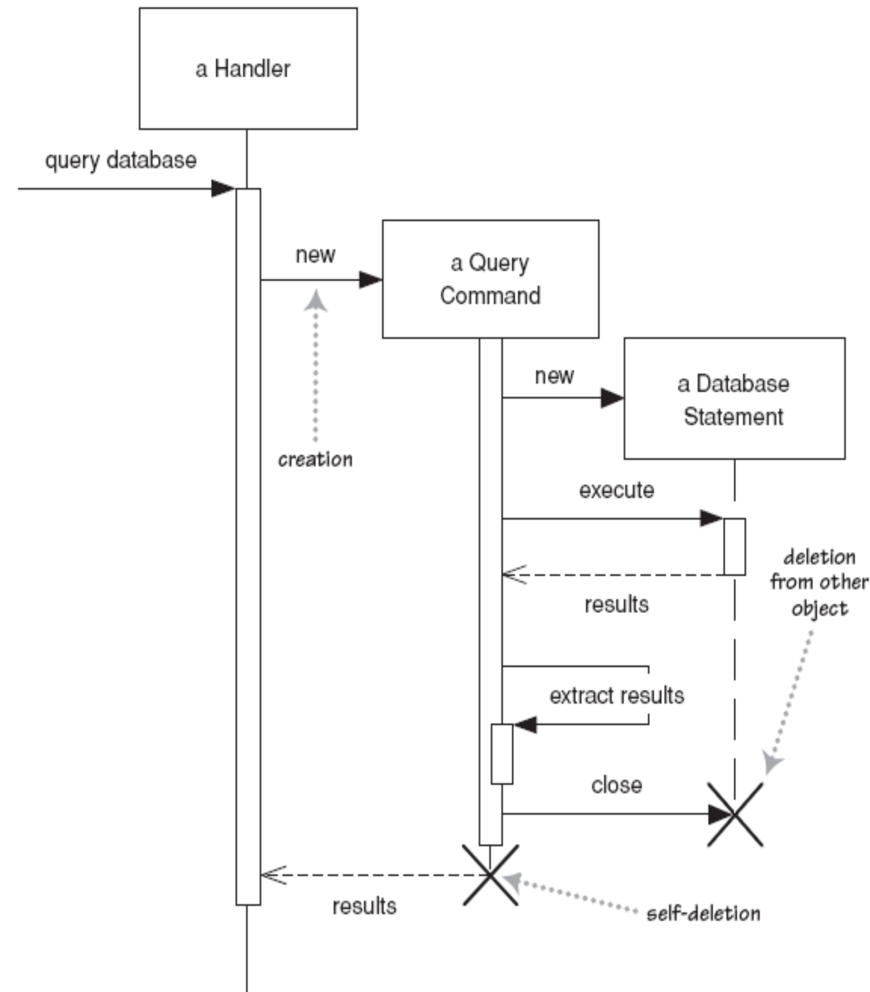
# Messages

- Message (method call) indicated by horizontal arrow to other object
  - dashed arrow back indicates return
  - different arrowheads for normal / concurrent (asynchronous) methods



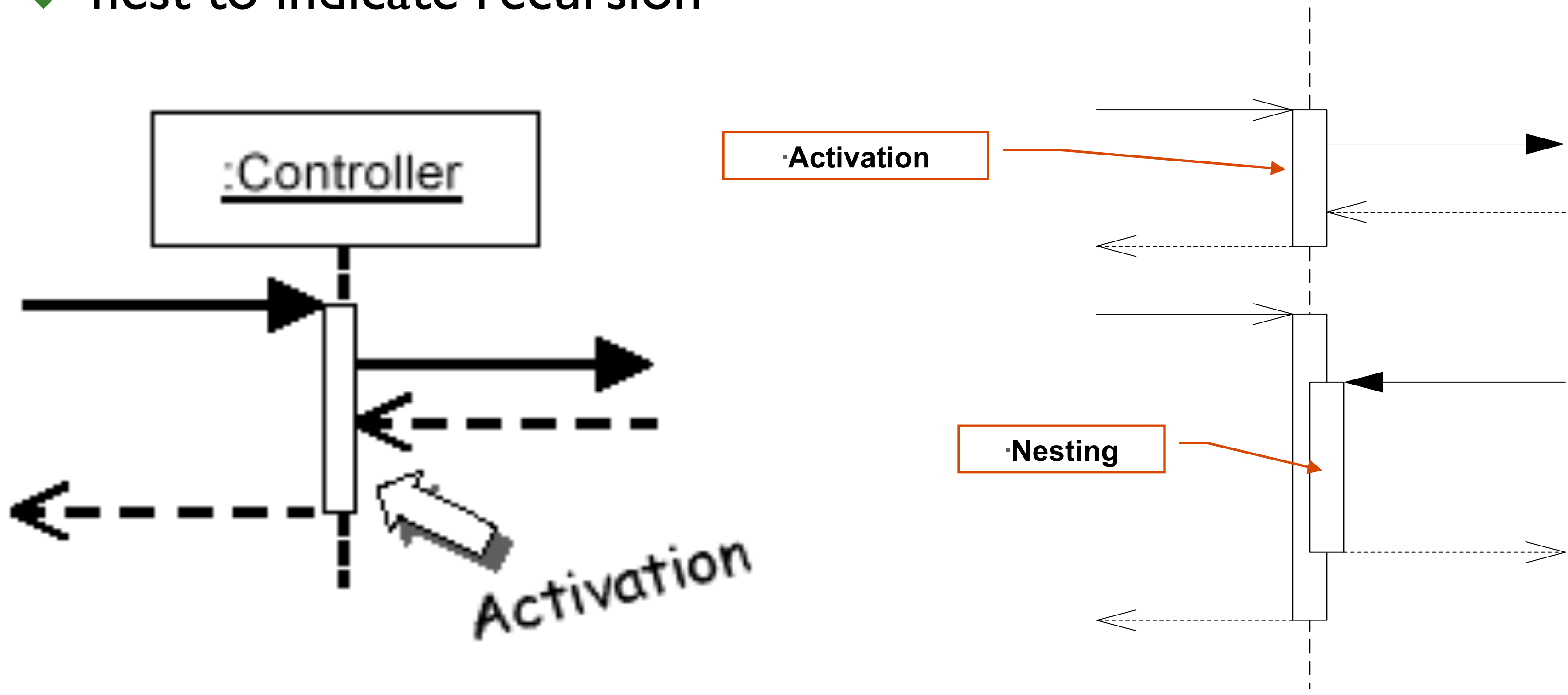
# Lifetime of Objects

- *Creation*: arrow with 'new' written above it
  - notice that an object created after the start of the scenario appears lower than the others
- *Deletion*: an X at bottom of object's lifeline
  - Java doesn't explicitly delete objects; they fall out of scope and are garbage-collected



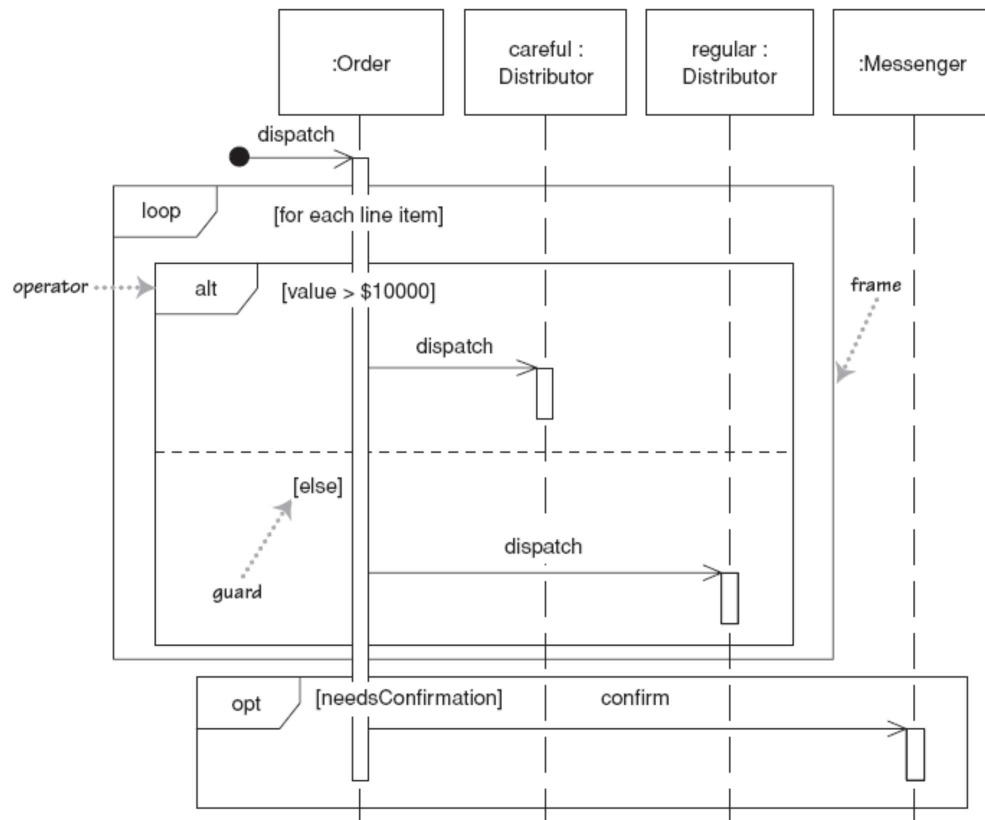
# Indicating Method Calls

- **Activation**: thick box over object's life line; drawn when object's method is on the stack
  - either that object is running its code, or it is on the stack waiting for another object's method to finish
  - nest to indicate recursion



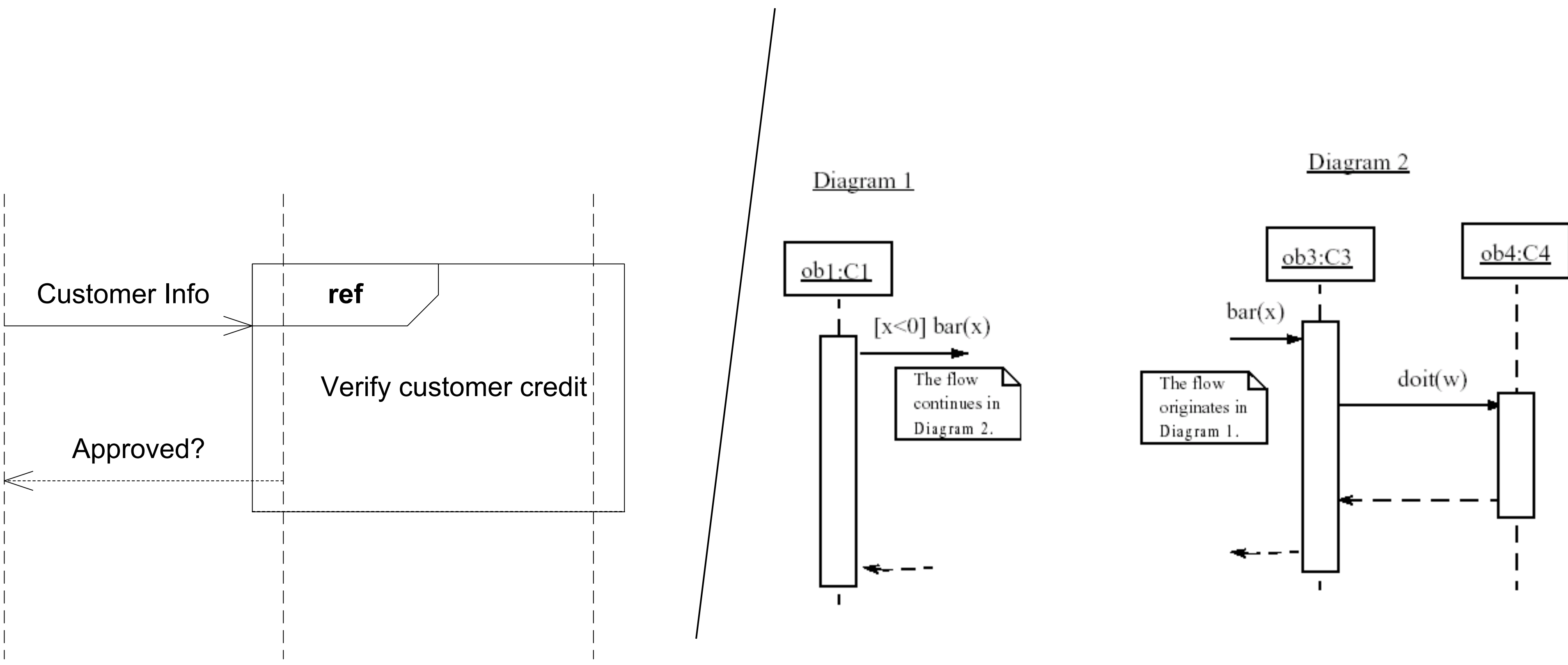
# Indicating Selection and Loops

- frame: box around part of a sequence diagram to indicate selection or loop
  - `if` -> (opt) [condition]
  - `if/else` -> (alt) [condition], separated by horizontal dashed line
  - loop -> (loop) [condition or items to loop over]



# Linking Sequence Diagrams

- If one sequence diagram is too large or refers to another diagram, indicate it with either:
  - an unfinished arrow and comment
  - a "ref" frame that names the other diagram



# Example – Shopping Cart

sd Example

StoreFront

Cart

Inventory

loop

AddItem

ReserveItem

Checkout

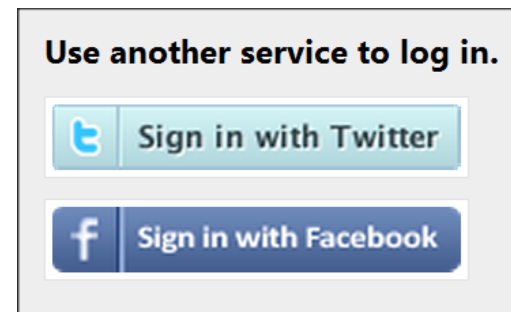
ProcessOrder

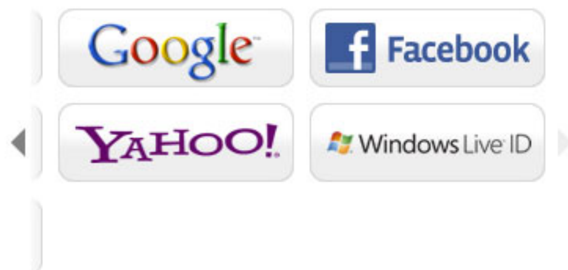
ConfirmOrder

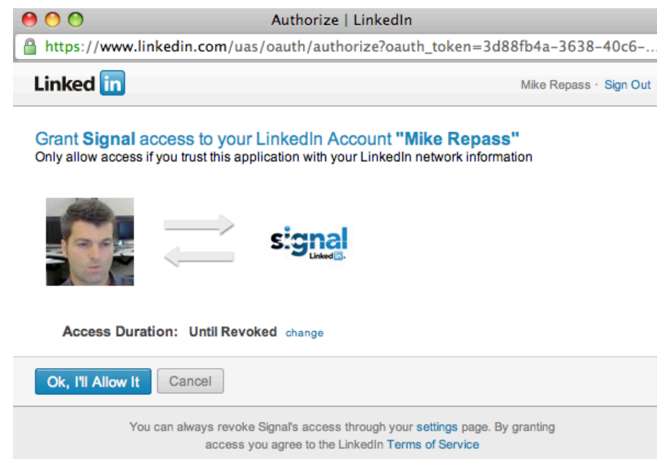
PlaceItemInOrder

# Example – Oauth 2.0

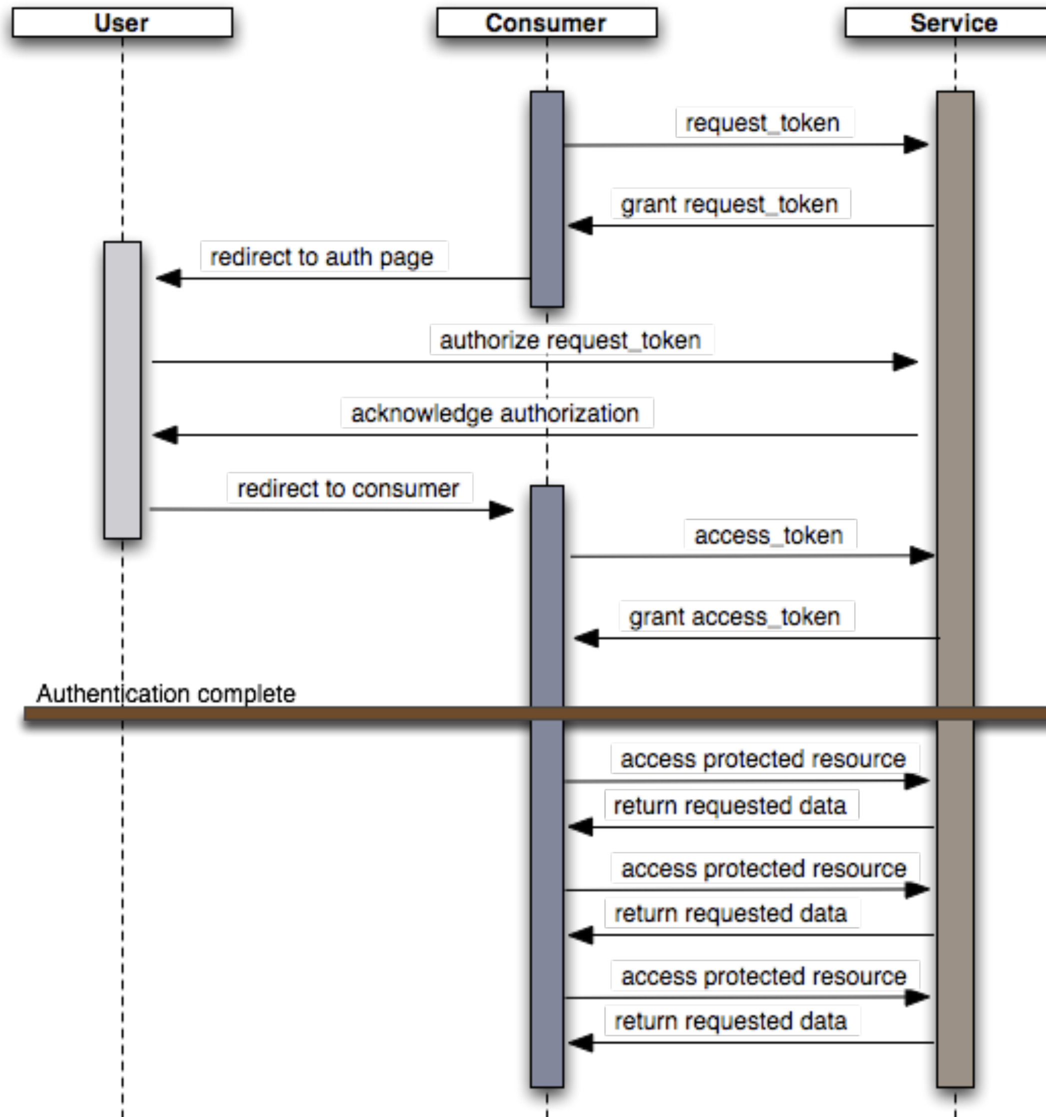
- OAuth provides client applications a secure delegated access to server resources on behalf of a resource owner





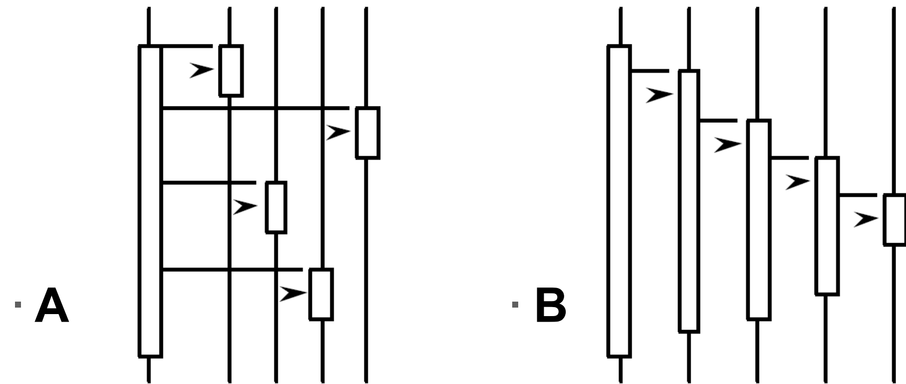



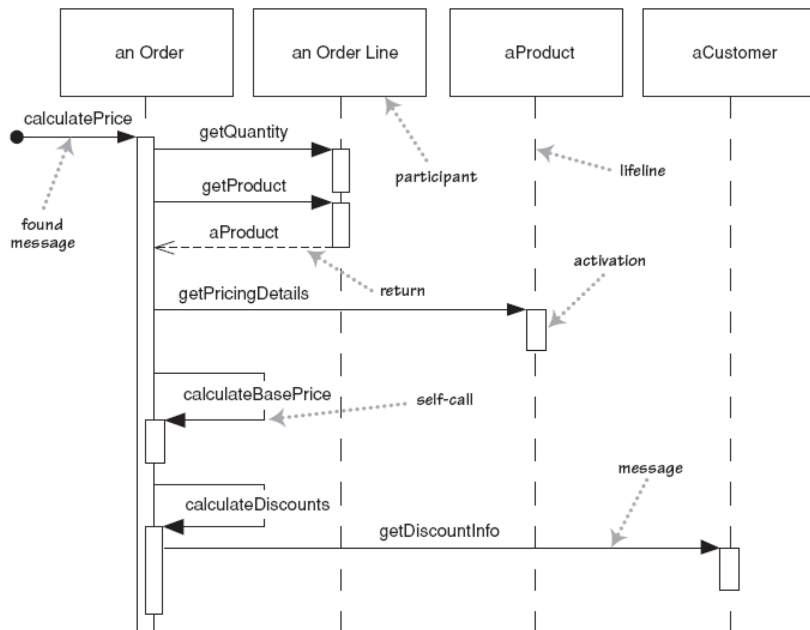
# Example – Oauth 2.0

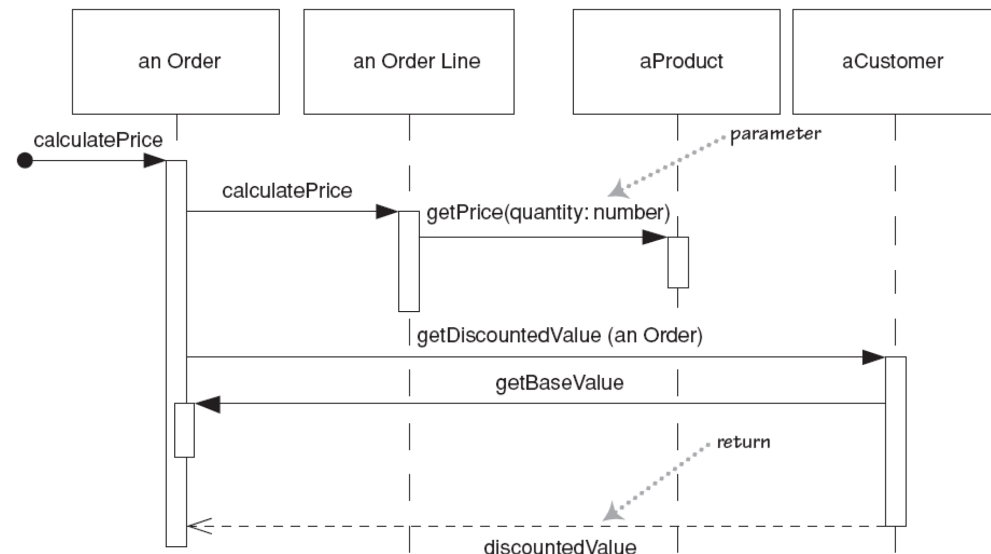


# Forms of System Control

- What can you say about the control flow of each of the following systems?
  - Is it centralized?
  - Is it distributed?



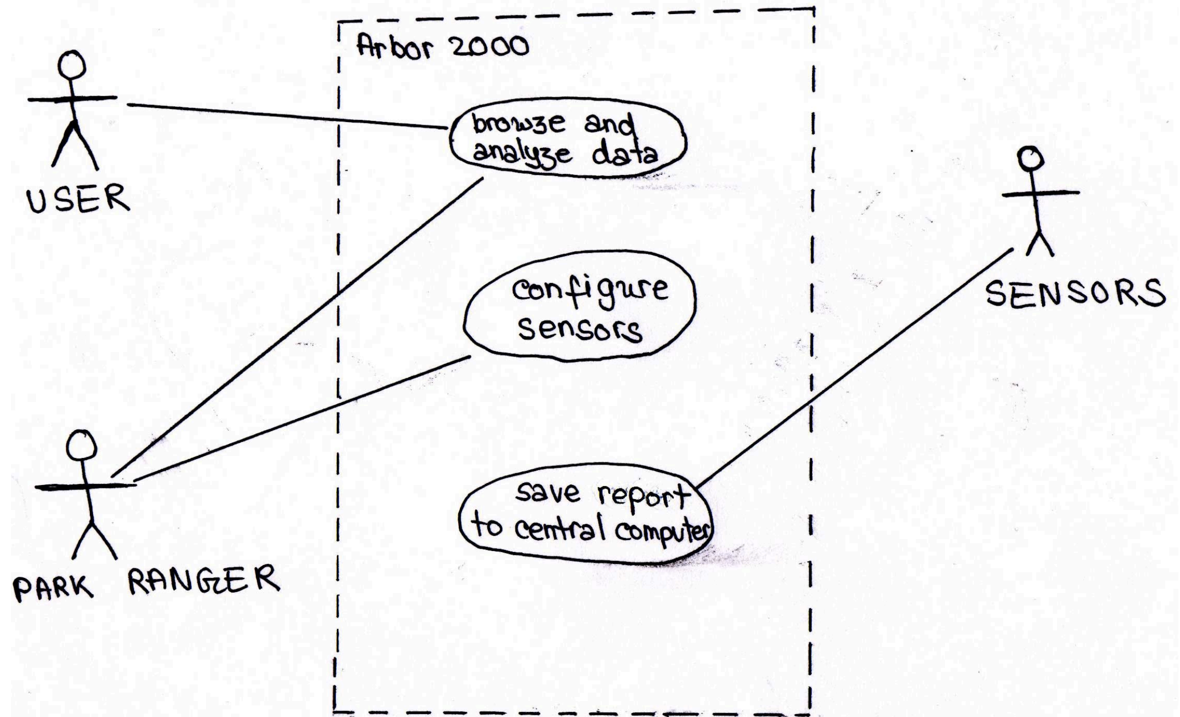



# Why not just code it?

- Sequence diagrams can be somewhat close to the code level. So why not just code up that algorithm rather than drawing it as a sequence diagram?
  - a good sequence diagram is still a bit above the level of the real code (not all code is drawn on diagram)
  - sequence diagrams are language-agnostic (can be implemented in many different languages
  - non-coders can do sequence diagrams
  - easier to do sequence diagrams as a team
  - can see many objects/classes at a time on same page (visual bandwidth)

# 3. Use Case Diagram



# Use Cases

- What is a Use Case?
  - A formal way of representing how a business system interacts with its environment
  - Illustrates the activities that are performed by the users of the system
  - A scenario-based technique in the UML
  - A sequence of actions a system performs that yields a valuable result for a particular actor

# Use Cases

- **Use case diagrams** describe what a system does from the standpoint of an external observer. The emphasis is on *what* a system does rather than *how*.

- Use case diagrams are closely connected to scenarios. A **scenario** is an example of what happens when someone interacts with the system.

# Use Cases

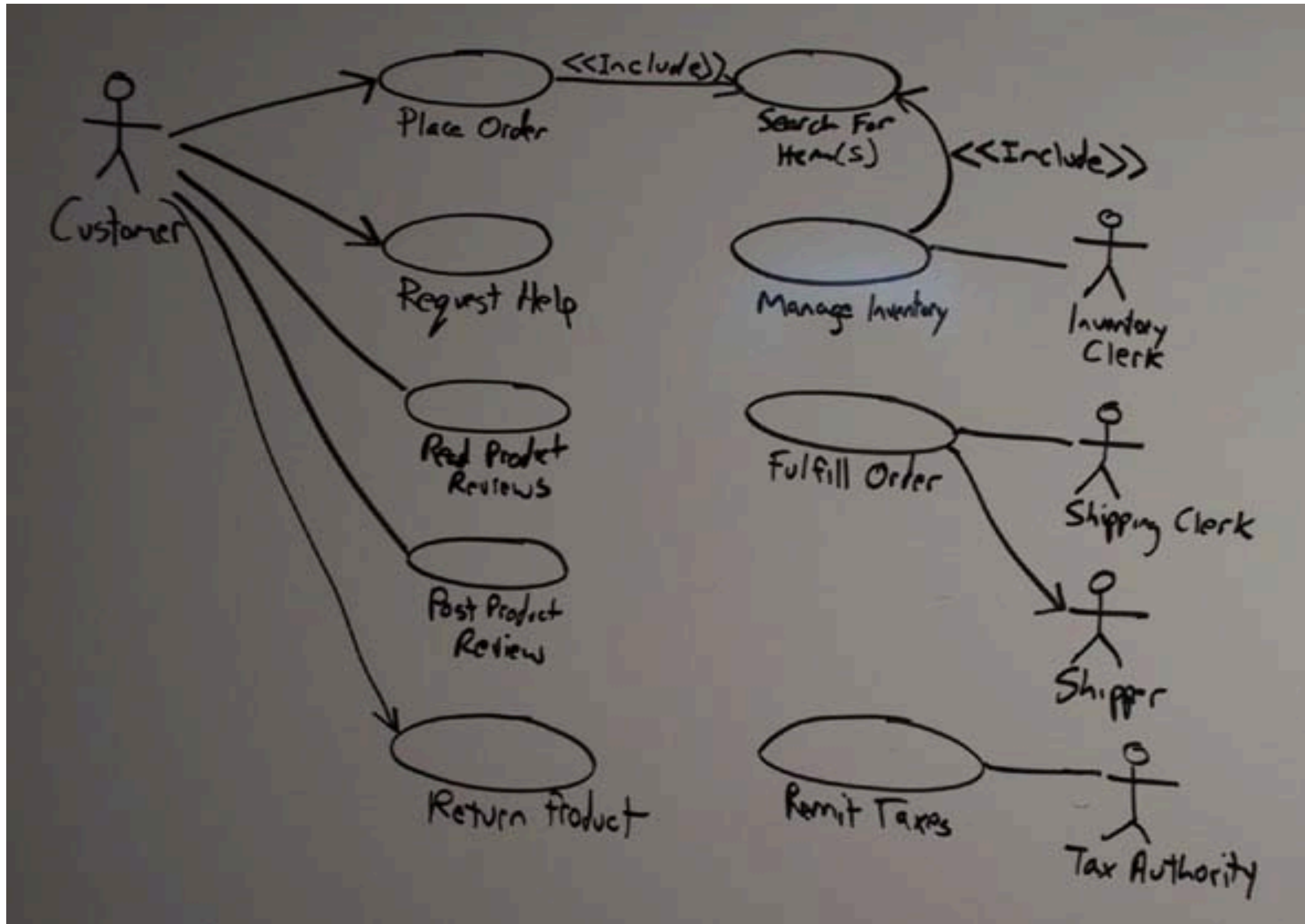
- Here is a scenario for a medical clinic.
  - A *patient calls the clinic to make an appointment for a yearly checkup. The receptionist finds the nearest empty time slot in the appointment book and schedules the appointment for that time slot.* "
- We want to write a use case for this scenario.
- Remember: A **use case** is a summary of scenarios for a single task or goal.

# Use Cases

- The picture below is a **Make Appointment** use case for the medical clinic.
- The actor is a **Patient**. The connection between actor and use case is a **communication association** (or **communication** for short).

**Actors** are stick figures.
**Use cases** are ovals.
**Communications** are lines that link actors to use cases.

# Use Case in Practice



# UML in the Industry Today

- Not as popular as in the 90's
- Important for us to learn OOP (e.g., design patterns)
- Helpful in communication and visualization
  - Class diagram
  - Sequence diagram
- White boarding sketching is more important than tool-based diagrams
- Could be asked during tech-Interview!