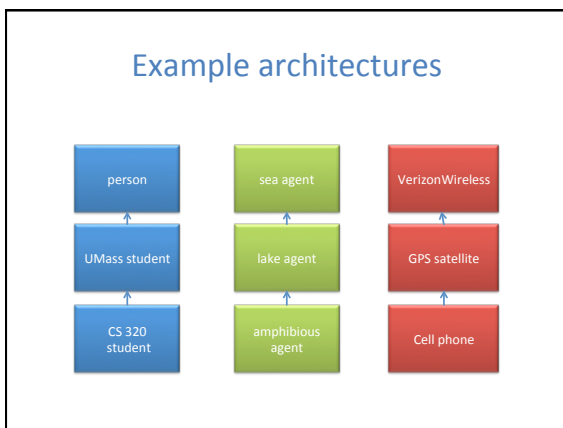


UML



Design and UML Diagrams

How do people  
draw / write down  
software architecture?



Big questions

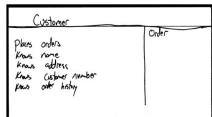
- What is UML?
  - Why should I bother? Do people really use UML?
- What is a UML class diagram?
  - What kind of information goes into it?
  - How do I create it?
  - When should I create it?

Design phase

- **design**: specifying the structure of how a software system will be written and function, without actually writing the complete implementation
- a transition from "what" the system must do, to "how" the system will do it
  - What classes will we need to implement a system that meets our requirements?
  - What fields and methods will each class have?
  - How will the classes interact with each other?

How do we design classes?

- class identification from project spec / requirements
  - nouns are potential classes, objects, fields
  - verbs are potential methods or responsibilities of a class
- CRC card exercises
  - write down classes' names on index cards
  - next to each class, list the following:
    - **responsibilities**: problems to be solved; short verb phrases
    - **collaborators**: other classes that are sent messages by this class (asymmetric)
- UML diagrams
  - class diagrams
  - sequence diagrams
  - ...



## UML

In an effort to promote Object Oriented designs, three leading object oriented programming researchers joined ranks to combine their languages:

- Grady Booch (BOOCH)
- Jim Rumbaugh (OML: object modeling technique)
- Ivar Jacobsen (OOSE: object oriented software eng)

and come up with an industry standard [mid 1990's].

## UML – Unified Modeling Language

- The result is large (as one might expect)
  - Union of all Modeling Languages
    - Use case diagrams
    - Class diagrams
    - Object diagrams
    - Sequence diagrams
    - Collaboration diagrams
    - Statechart diagrams
    - Activity diagrams
    - Component diagrams
    - Deployment diagrams
    - .....
  - But it's a nice standard that has been embraced by the industry.

## Introduction to UML

- UML: pictures of an OO system
  - programming languages are not abstract enough for OO design
  - UML is an open standard; lots of companies use it
- What is legal UML?
  - a *descriptive* language: rigid formal syntax (like programming)
  - a *prescriptive* language: shaped by usage and convention
  - it's okay to omit things from UML diagrams if they aren't needed by team/supervisor/instructor

## Uses for UML

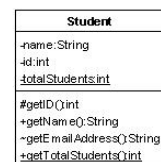
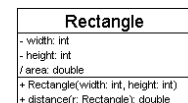
- as a sketch: to communicate aspects of system
  - forward design: doing UML before coding
  - backward design: doing UML after coding as documentation
  - often done on whiteboard or paper
  - used to get rough selective ideas
- as a blueprint: a complete design to be implemented
  - sometimes done with CASE (Computer-Aided Software Engineering) tools
- as a programming language: with the right tools, code can be auto-generated and executed from UML
  - only good if this is faster than coding in a "real" language

## UML class diagrams

- What is a UML class diagram?
  - **UML class diagram:** a picture of
    - the classes in an OO system
    - their fields and methods
    - 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

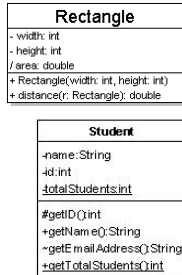
## Diagram of one class

- 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



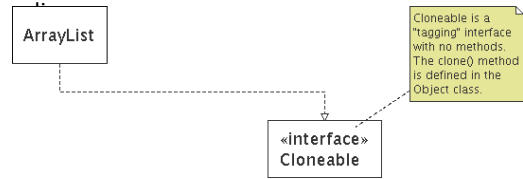
## Class attributes

- 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



## Comments

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

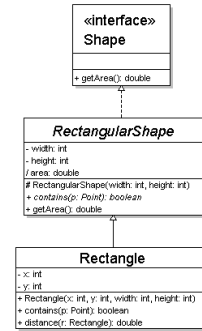


## Relationships between classes

- generalization:** an inheritance relationship
  - inheritance between classes
  - interface implementation
- association:** a usage relationship
  - dependency
  - aggregation
  - composition

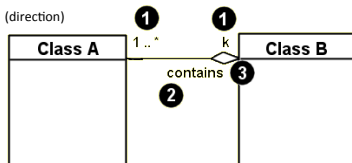
## Generalization relationships

- generalization (inheritance) relationships
  - 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



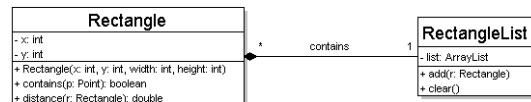
## Associational relationships

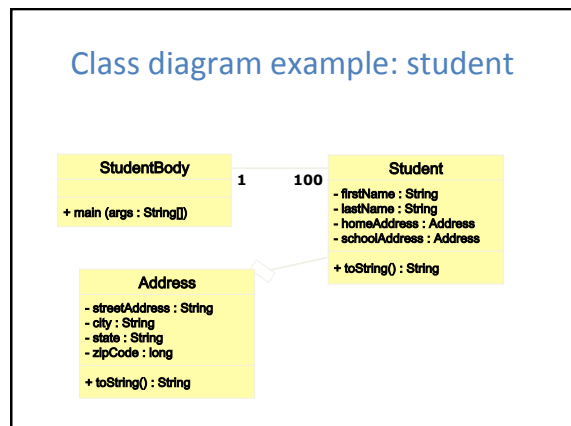
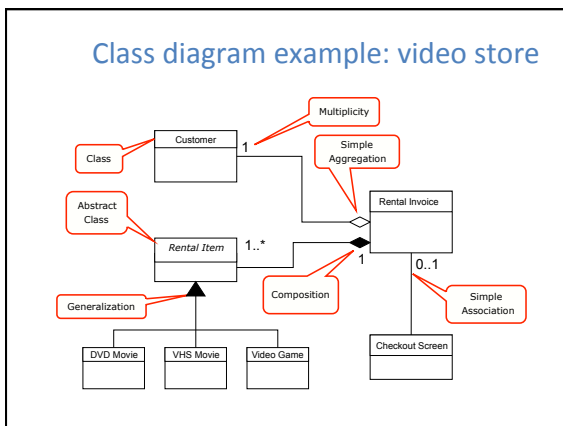
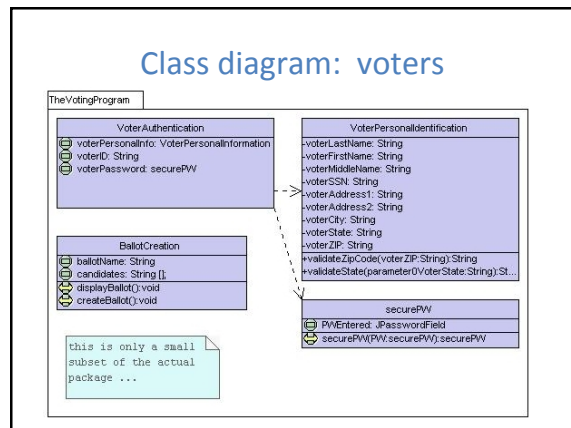
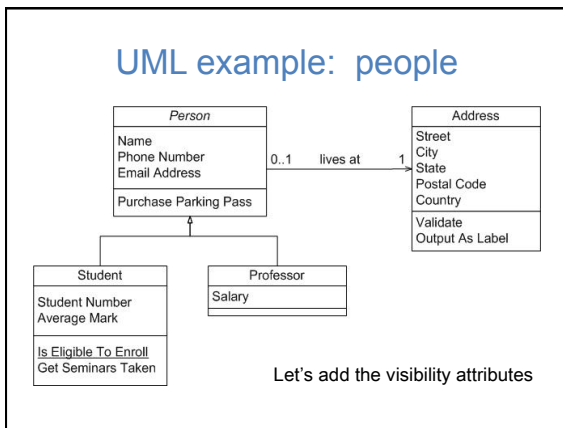
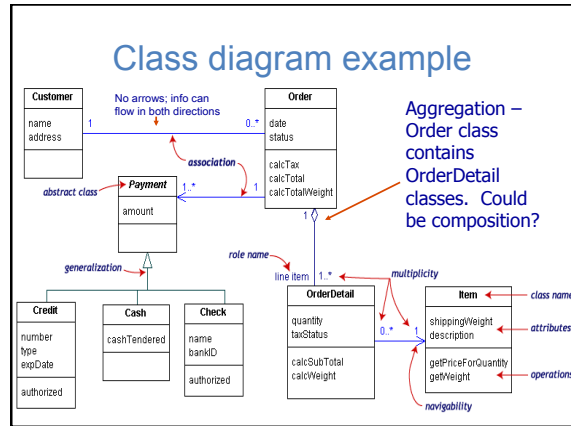
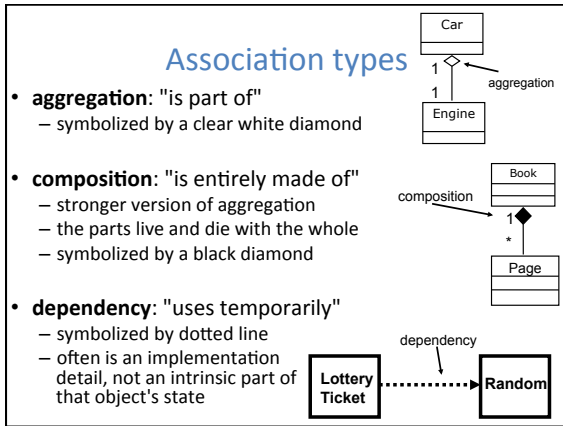
- associational (usage) relationships
  - multiplicity (how many are used)
    - \* ⇒ 0, 1, or more
    - 1 ⇒ 1 exactly
    - 2..4 ⇒ between 2 and 4, inclusive
    - 3..\* ⇒ 3 or more
  - name (what relationship the objects have)
  - navigability (direction)



## Multiplicity of associations

- one-to-one**
  - each student must carry exactly one ID card
- one-to-many**
  - one rectangle list can contain many rectangles





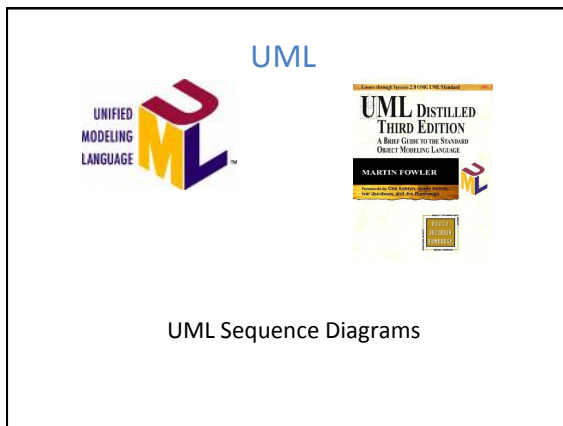
## Tools for creating UML diagrams

- Violet (free)
  - <http://horstmann.com/violet/>
- Rational Rose (trial)
  - <http://www.rational.com/>
- Visual Paradigm UML Suite (trial)
  - <http://www.visual-paradigm.com/>
  - direct download link: <https://www.visual-paradigm.com/download/>

(there are many others, but many are commercial)

## Class design exercise

- Consider this Texas Hold 'em poker game system:
  - 2 to 8 human or computer players
  - Each player has a name and stack of chips
  - Computer players have a difficulty setting: easy, medium, hard
  - Summary of each round:
    - 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.
  - What classes are in this system?  
What are their responsibilities?  
Which classes collaborate?
  - Draw a class diagram for this system. Include relationships between classes (generalization and associational).



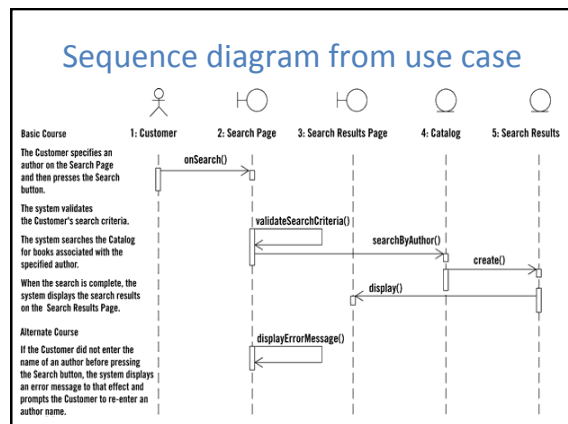
UML Sequence Diagrams

## 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)
- relation of UML diagrams to other exercises:
  - CRC cards → class diagram
  - use cases → sequence diagrams

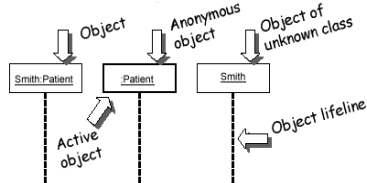
## Key parts of a sequence diagram

- **participant:** an object or an entity; the sequence diagram actor
  - sequence diagram starts with an unattached "found message" arrow
- **message:** communication between objects
- the axes in a sequence diagram:
  - horizontal: which object/participant is acting
  - vertical: time ( ↓ forward in time)



### Representing objects

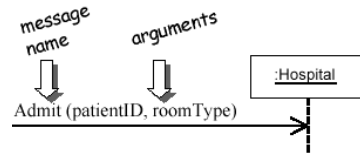
- An object: a square with object type, optionally preceded by object name and colon
  - write object's name if it clarifies the diagram
  - object's "life line" represented by dashed vert.



Name syntax: <objectname>.<classname>

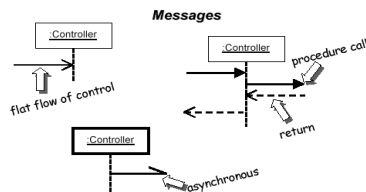
### Messages between objects

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



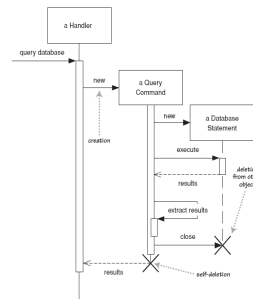
### Different types of messages

- Type of arrow indicates types of messages
  - dashed arrow back indicates return
  - different arrowheads for normal / concurrent (asynchronous) methods



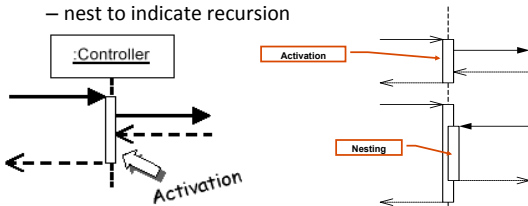
### Lifetime of objects

- **creation:** arrow with 'new' written above it
  - 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



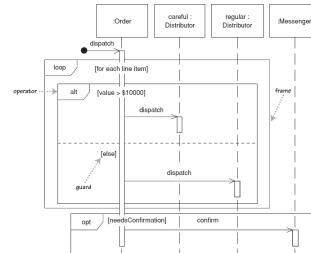
### Indicating method calls

- **activation:** thick box over object's life line
  - Either: that object is running its code or it is on the stack waiting for another object's method
  - nest to indicate recursion



### Selection and loops

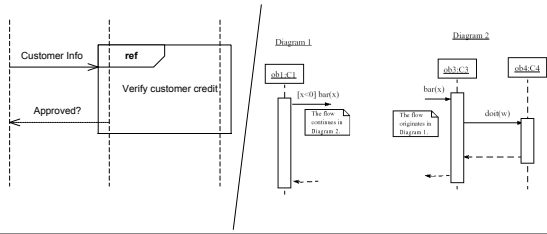
- frame: box around part of a sequence diagram to indicate selection or loop
  - if -> (opt) [condition]
  - if/else -> (alt) [condition], separated by horizontal dashed line
  - loop -> (loop) [condition or items to loop over]



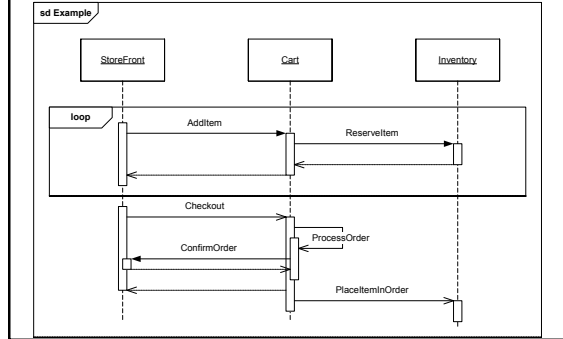
### Linking sequence diagrams

If one sequence diagram is too large or refers to another diagram:

- an unfinished arrow and comment
- a "ref" frame that names the other diagram



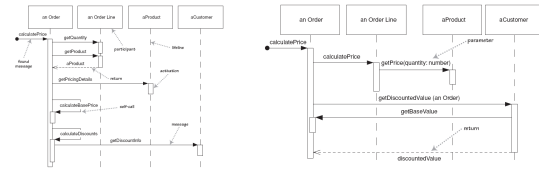
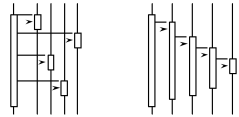
### Example sequence diagram



### 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)

### Poker sequence diagram exercise

The scenario begins when the player chooses to start a new round in the UI. The UI asks whether any new players want to join the round; if so, the new players are added using the UI.

All players' hands are emptied into the deck, which is then shuffled. The player left of the dealer supplies a blind bet of the proper amount. Next, each player is dealt a hand of two cards from the deck in a round-robin fashion; one card to each player. Then the second card.

If the player left of the dealer doesn't have enough money for his/her blind, he/she is removed from the game and the next player supplies the blind. If that player also cannot afford the blind, this cycle continues until a rich-enough player is found or all players are removed.

### Calendar sequence diagram exercise

The user chooses to add a new appointment in the UI. The UI notices which part of the calendar is active and pops up an Add Appointment window for that date and time.

The user enters the necessary information about the appointment's name, location, start and end times. The UI will prevent the user from entering an appointment that has invalid information, such as an empty name or negative duration. The calendar records the new appointment in the user's list of appointments. Any reminder selected by the user is added to the list of reminders.

If the user already has an appointment at that time, the user is shown a warning message and asked to choose an available time or replace the previous appointment. If the user enters an appointment with the same name and duration as an existing group meeting, the calendar asks the user whether he/she intended to join that group meeting instead. If so, the user is added to that group meeting's list of participants.