CSE 403 Autumn 2012

UML

Class and Sequence Diagrams

Anton Osobov

Slides adapted from Marty Stepp, CSE 403, Winter 2012

Reminders

- Architecture doc(s) due tonight at 11:59 PM
- Presentations on 10/24
- Quiz on 10/26

Design Phase

- **design**: Specifying the structure of how a software system will be written and function (without actually writing the code).

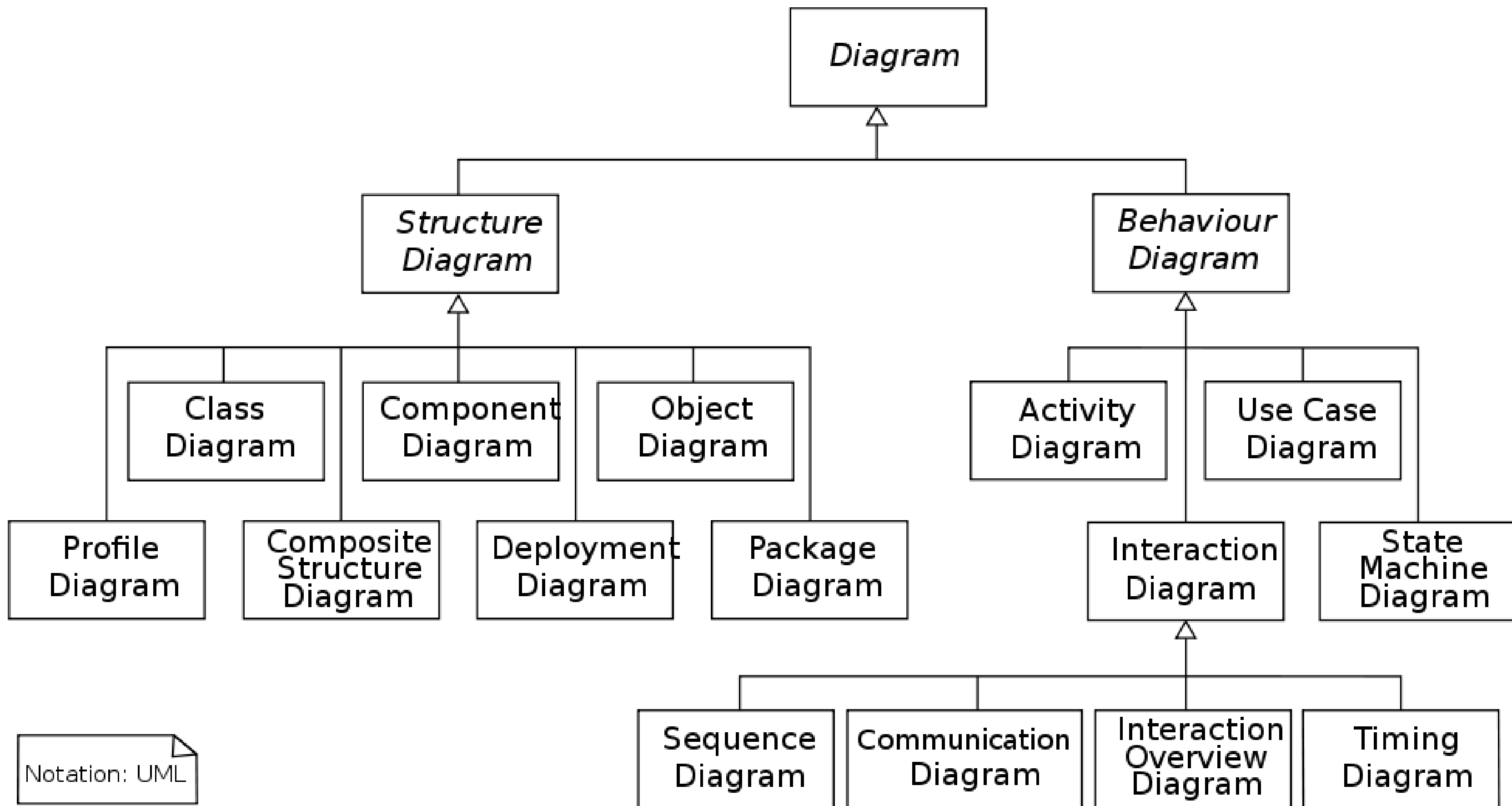
- a transition from "what" the system must do, to "how" the system will do it
  - What classes will we need in order to implement a system that meets our requirements?
  - What fields and methods will each class have?
  - How will the classes interact with each other?

Introduction to UML

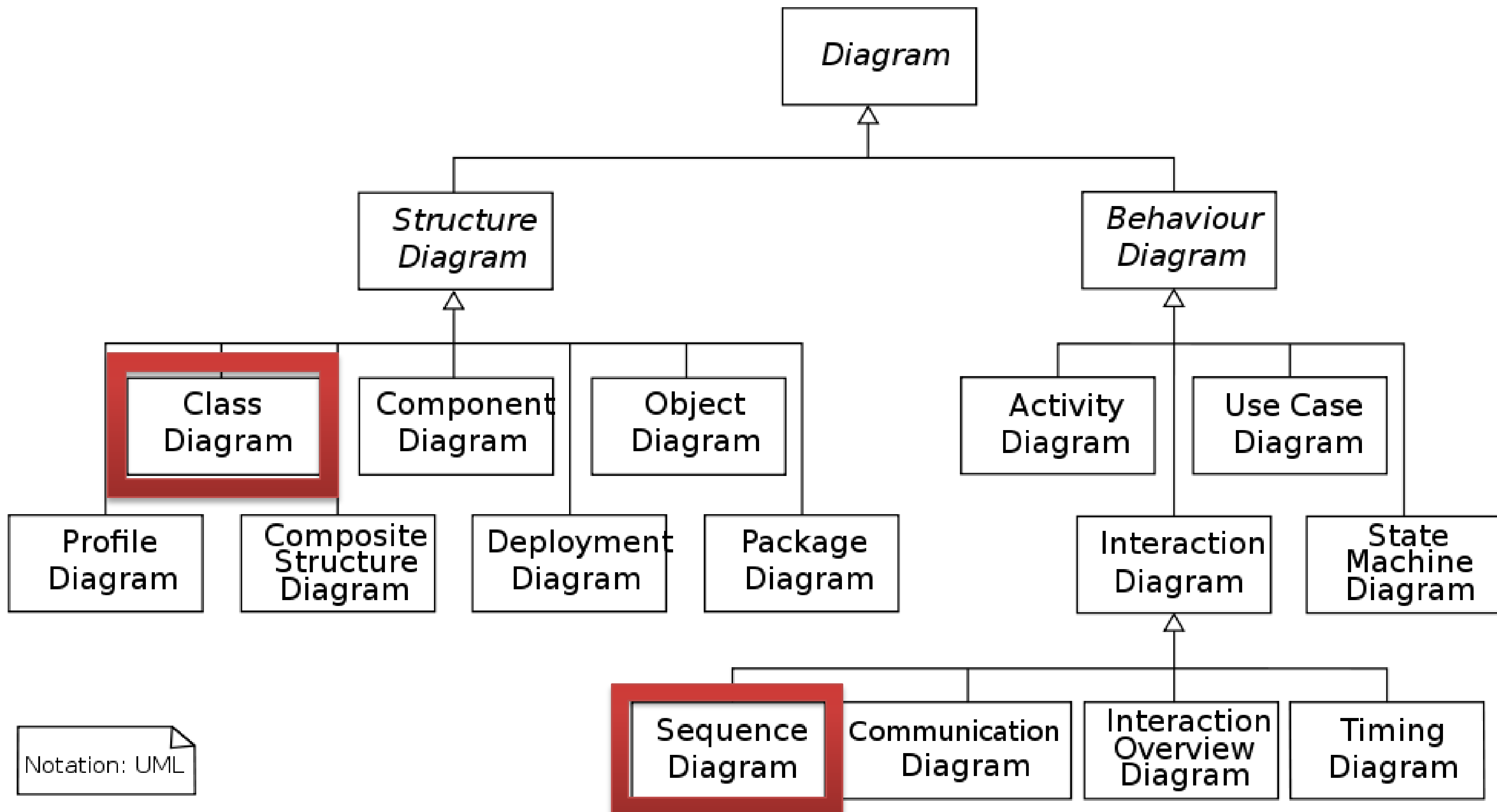
- **Unified Modeling Language (UML)**: depicts an OO system
  - programming languages are not abstract enough for OO design
  - UML is an open standard; lots of companies use it
    - many programmers either know UML or a "UML-like" variant

- UML is ...
  - a *descriptive* language: rigid formal syntax (like programming)
  - a *prescriptive* language: shaped by usage and convention
  - UML has a rigid syntax, but some don't follow it religiously
  - it's okay to omit things from UML diagrams if they aren't needed by team/supervisor/instructor

Introduction to UML



Introduction to UML



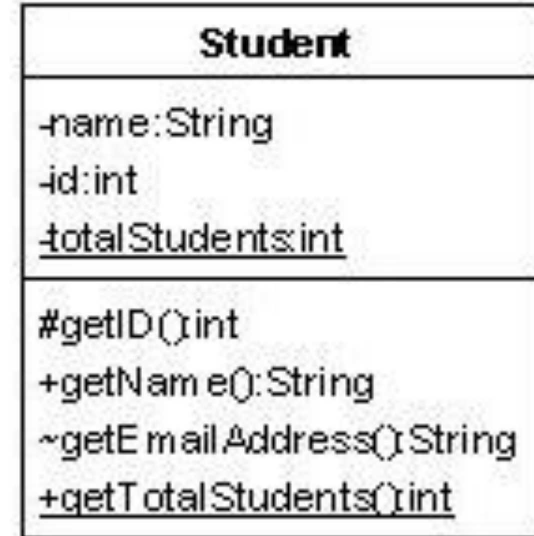
UML Class Diagrams

- **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

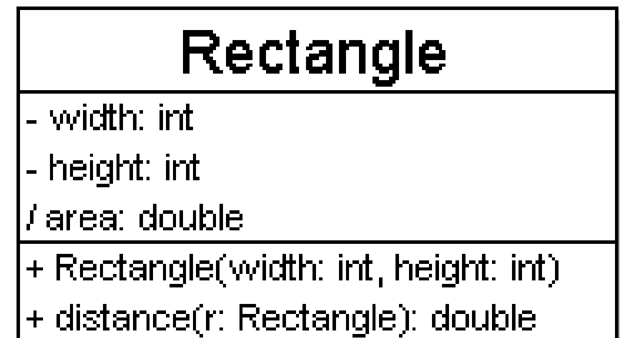
- What are some things <u>not</u> represented in a class diagram?
  - details of how the classes interact
  - algorithmic details; how particular behavior is implemented
  - trivial methods (`get`/`set`)
  - classes that come from libraries (`ArrayList`, etc.)

Diagram of a class

- **class** name in top of box
  - write <<interface>> on top of interface's names
  - use *italics* for an abstract class name



- **attributes**
  - include all fields of the object
  - Include "derived" properties
- **operations** (constructors/methods)
  - may omit trivial methods – get/set
    - except from an interface
  - should not include inherited methods



Class attributes

- syntax:
  - visibility name : type [count] = defaultValue

Symbol	Visibility
+	public
#	protected
-	private
~	package (default)
/	derived

- underline <u>static attributes</u>

Student

-name:String
-id:int
<u>-totalStudents:int</u>

#getID():int
+getName():String
~getEmailAddress():String
<u>+getTotalStudents():int</u>

Rectangle

- width: int
- height: int
/ area: double

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

Class operations/methods

- syntax:
  - visibility name(parameters): returnType

- underline static methods
- parameter types listed as (name: type)
- omit *returnType* on constructors and when return is `void`

Student

-name:String
-id:int
-totalStudents:int

#getID():int
+getName():String
~getEmailAddress():String
+getTotalStudents():int

Rectangle

- width: int
- height: int
/ area: double

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

Relationships between classes

- **generalization**: an inheritance relationship
  - inheritance between classes
  - interface implementation

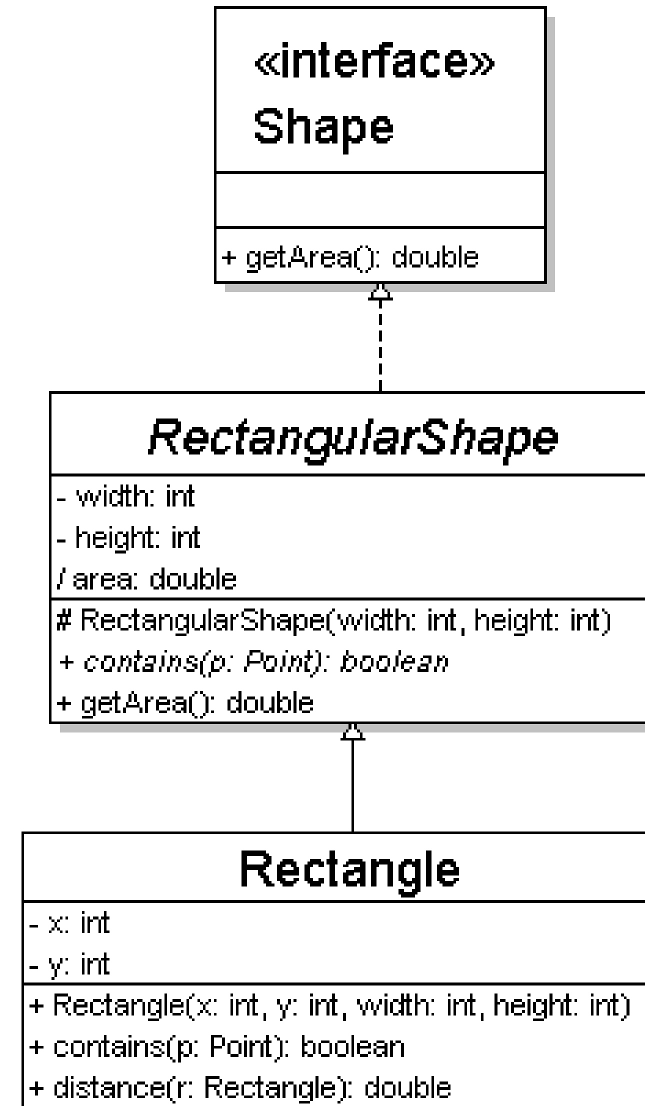
- **association**: a usage relationship
  - dependency
  - aggregation
  - composition

Generalization

- hierarchies are drawn top down
  - arrow from child to parent

Parent	Line/Arrow Style
class	solid, black arrow
abstract class	solid, white arrow
interface	dashed, white arrow

- trivial/obvious relationships often not drawn
  - Java: Object

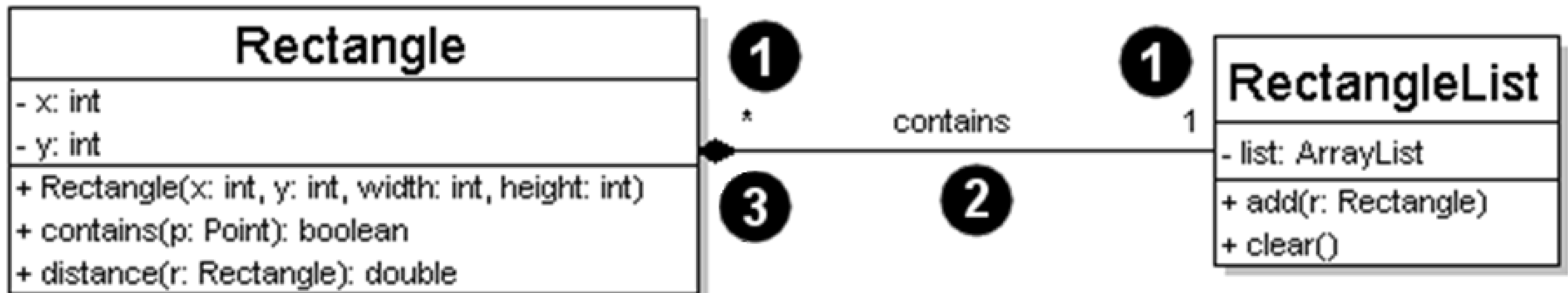


Association

1. multiplicity

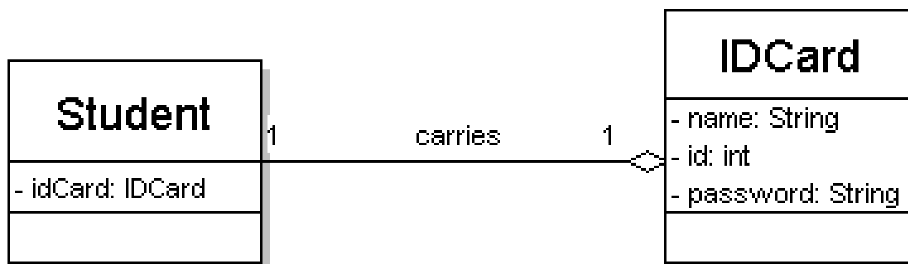
Symbol	How many are used?
*	0, 1, or more
1	exactly 1
2..4	between 2 and 4
5..*	5 or more

2. name
   - what relationship the objects have
3. navigability
   - direction

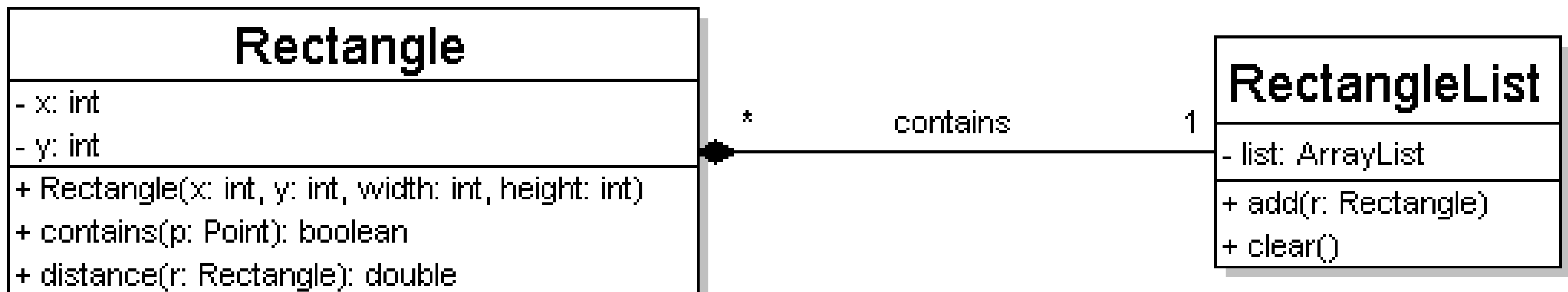


Multiplicity

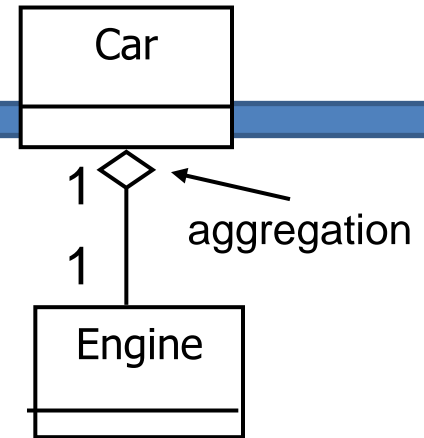
- one-to-one
  - each student must have exactly one ID card



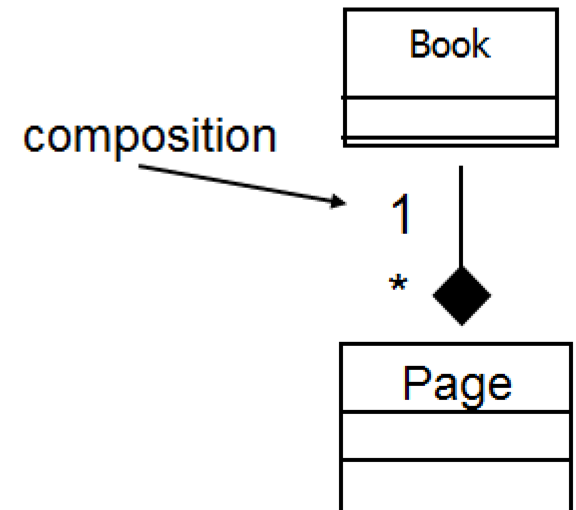
- one-to-many
  - a RectangleList can contain 0, 1, 2, ... rectangles



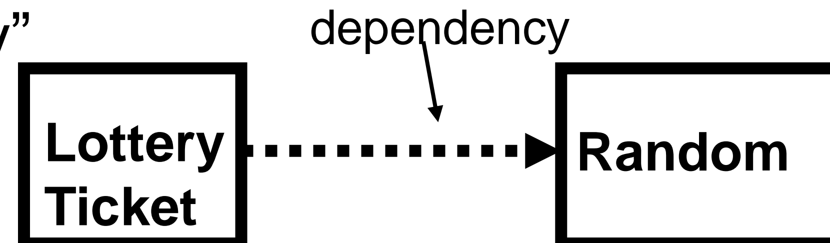
Association types



- **aggregation:** "is part of"
  - clear, white diamond



- **composition:** "is entirely made of"
  - stronger version of aggregation
  - the parts only exist while the whole exists
  - black diamond



- **dependency:** "uses temporarily"
  - dotted arrow or line

UML Sequence Diagrams

- **sequence diagram:** an “interaction diagram” that models a single scenario executing in the system

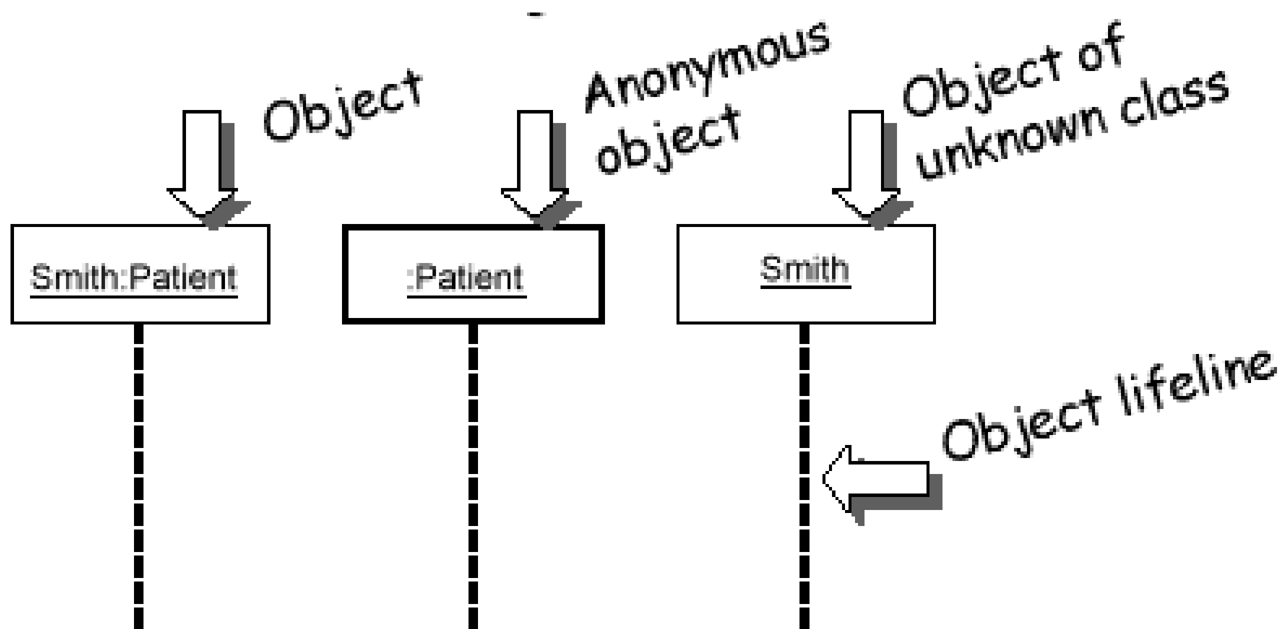
- UML representation of a use case

Sequence diagram key parts

- **participant:** object or entity that acts in the diagram
- **message:** communication between participants
- axes in a sequence diagrams
  - horizontal: which participant/object is acting
  - vertical: time (down = forward in time)

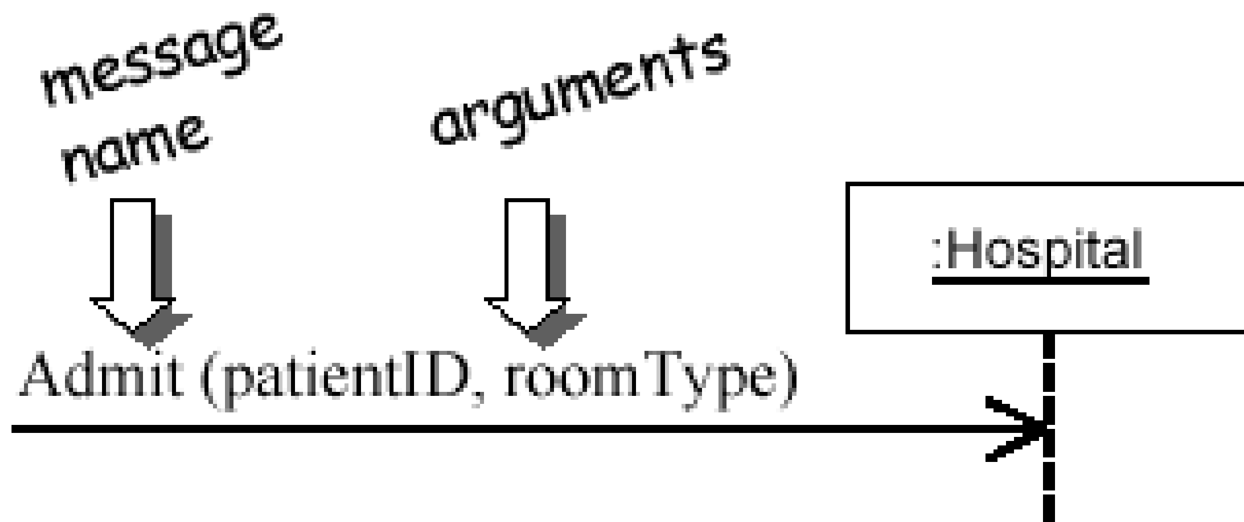
Representing objects

- Rectangles with object type, optionally preceded by “name : ”
  - Write object’s name if it clarifies the diagram
  - Object’s “life line” represented by dashed vertical line



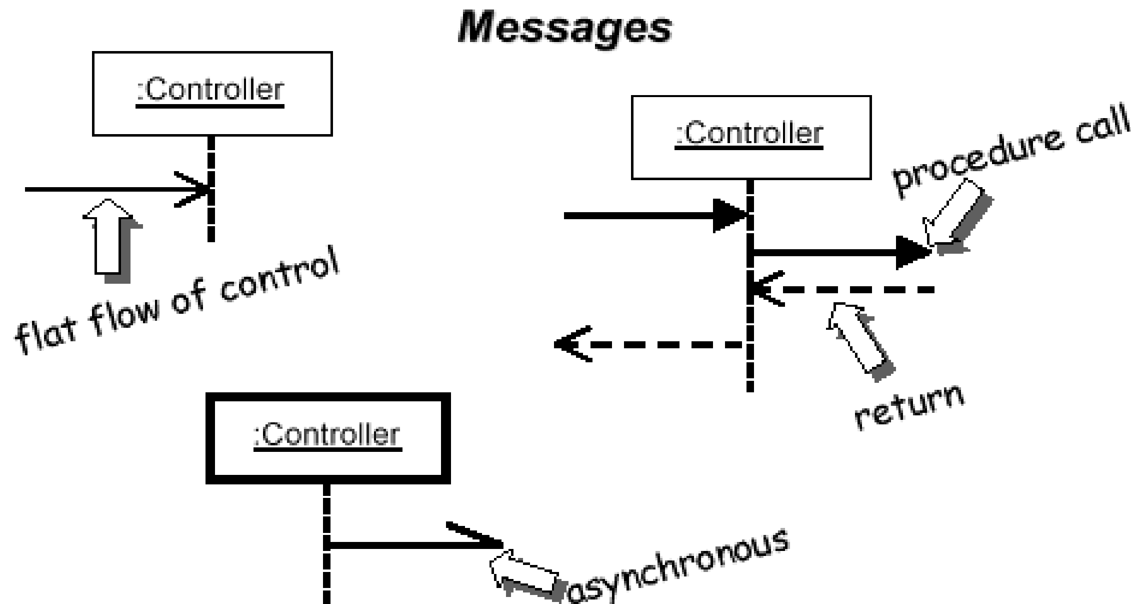
Messages between objects

- messages (methods calls) represented by arrow to other object
  - method name and arguments written above the arrow



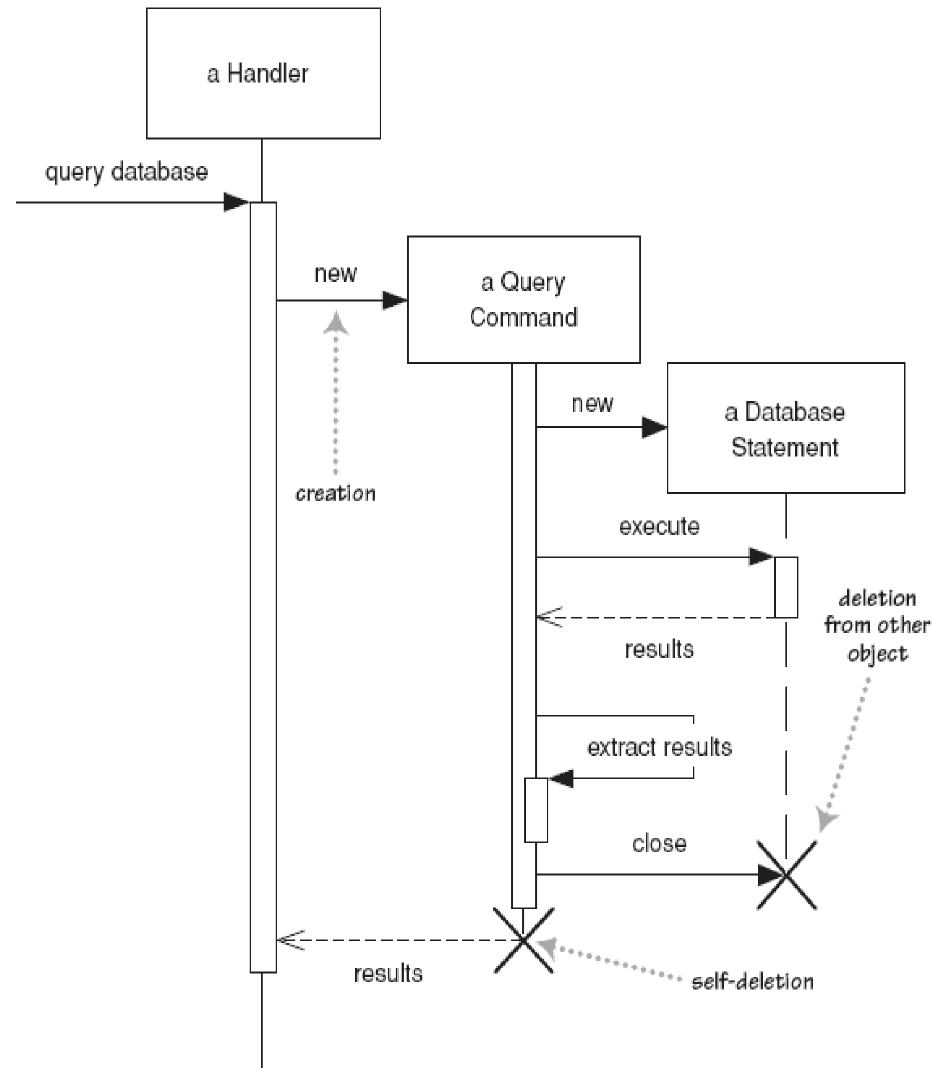
Messages continued

- messages (methods calls) represented by arrow to other object
  - dashed arrow back indicates return
  - different arrows for normal and concurrent/asynchronous calls



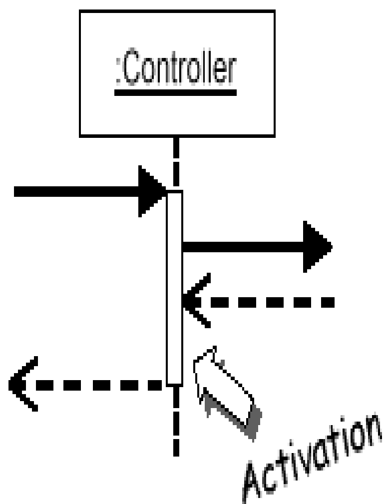
Object lifetime

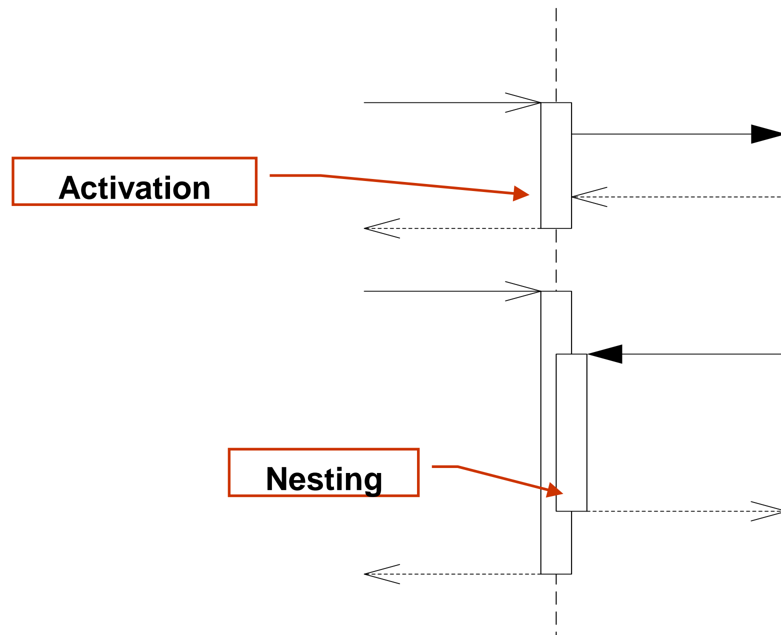
- *creation:* arrow with “new” written above it
  - object created after start of scenario appears lower than the others
- *deletion:* an X at bottom of object’s lifeline
  - more applicable to languages with manual memory management (C, C++)



Method activiation

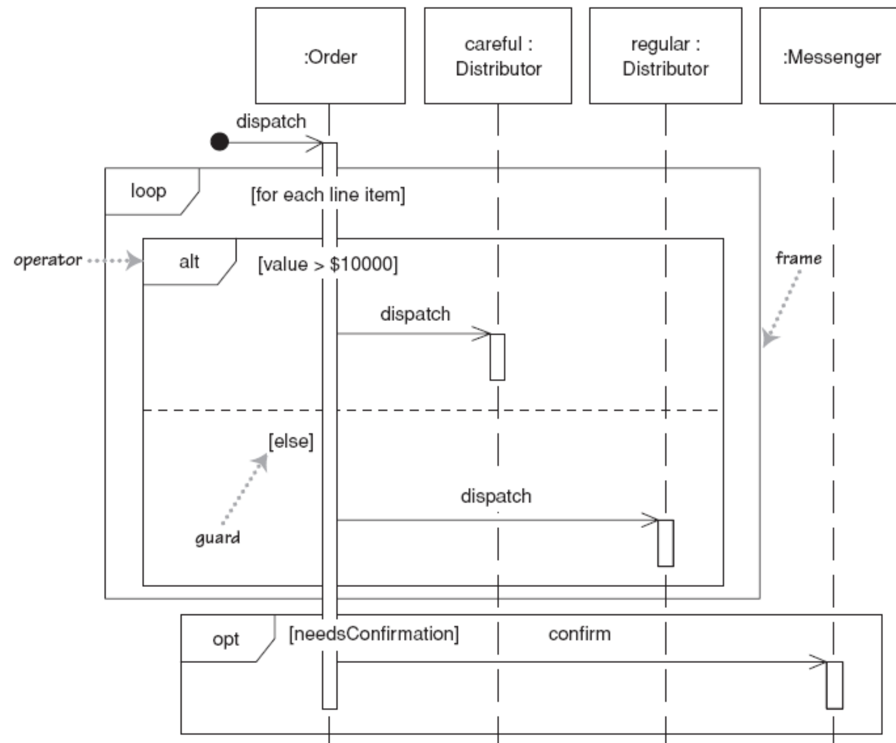
- **activation:** thick box over object's life line; drawn when object's method is on the stack
  - either that object is running its code, or it is on the stack waiting for another object's method to finish
  - nest activations to indicate recursion





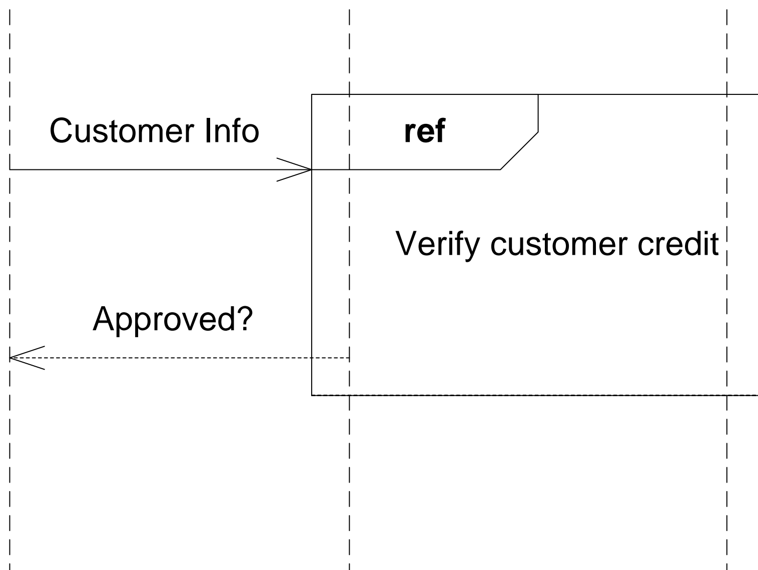
If statements and loops

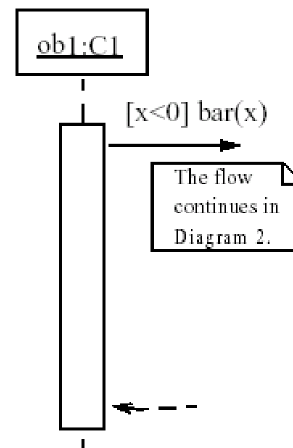
- **frame**: box around part of diagram to indicate `if` or loop
  - `if` -> (opt) [condition]
  - `if/else`-> (alt) [condition], separated by horizontal dashed line
  - loop -> (loop) [condition or items to loop over]

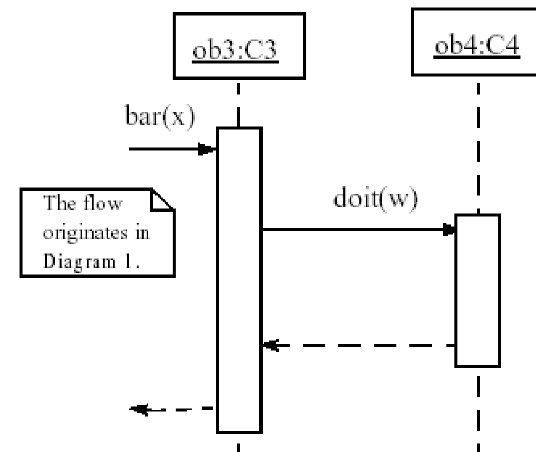


Linking sequence diagrams

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







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?

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)

Tools for creating UML

- Free
  - Violet
  - Dia
  - UMLet
  - ArgoUML
- Not so free
  - IBM Rational Rose
  - Visual Paradigm UML Suite

Questions?