Computer Science
UNIVERSITY OF TORONTO

lecture 1: introduction to modeling & UML

csc302h
winter 2014

CUSEC 2014

canadian university software engineering conference

january 16 – 18, montreal
http://2014.cusec.net

SOLD OUT

administrative

- assignment #1 out by tuesday
- form groups today
- sign up on piazza asap!
  - any problems?
  - everyone familiar with piazza?

recap from last time

- engineering large software systems is difficult!
  - $bn wasted annually on botched projects
  - it isn't just the big ones that go awry (see boyd's toast), but they tend to with a greater probability

- for our purposes, "large" means anything non-trivial that benefits from proper planning and tools, and is likely to be used by someone other than the developer

recap from last time (2)

- work will be done in teams of 6-7
  - initial groups will be formed today in the tutorial hour.
- we will be working on a large open source project
  - project(s) selection will be finalized on tuesday when a1 goes out.

modeling

- one thing that we as software developers/ engineers can do to better understand software is by using models
- many choices when building models
  - multiple modeling “languages”
  - graphical/Textual
  - diagrams – ER diagrams for data, class and object diagrams in OOP.
  - ad-hoc
- for this course we’ll use UML (more or less)

modeling (2)

- uml as defined by wikipedia:

  “UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. The UML includes a set of graphic notation techniques to create visual models of object-oriented software-intensive systems.”

- caveat: how often do I use (strict) uml?

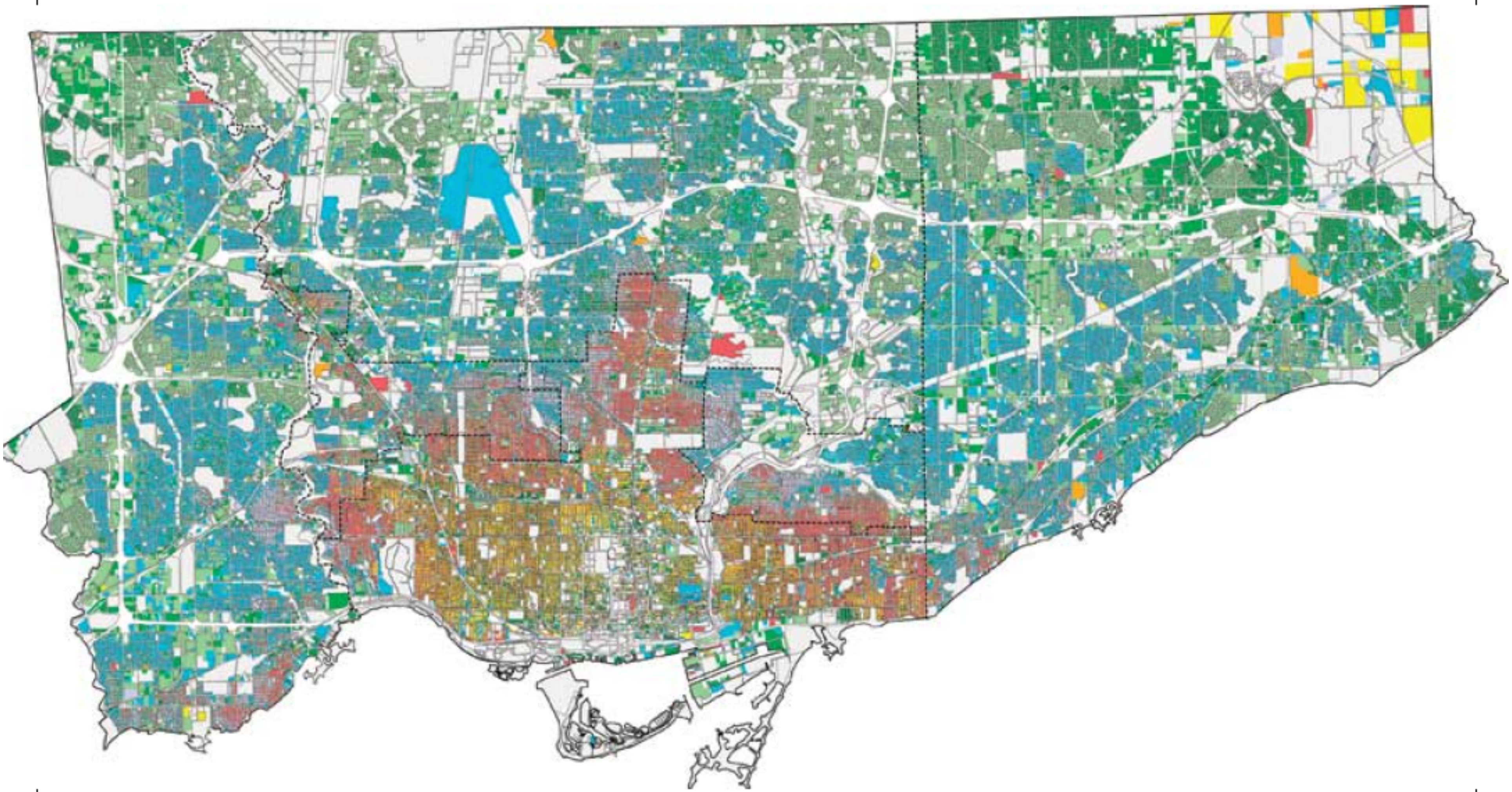
  “…in his eighteen years as a professional programmer, Wilson had only ever worked with one programmer who actually used it voluntarily .” – Two Solitudes Illustrated, Greg Wilson & Jorge Aranda, 2012

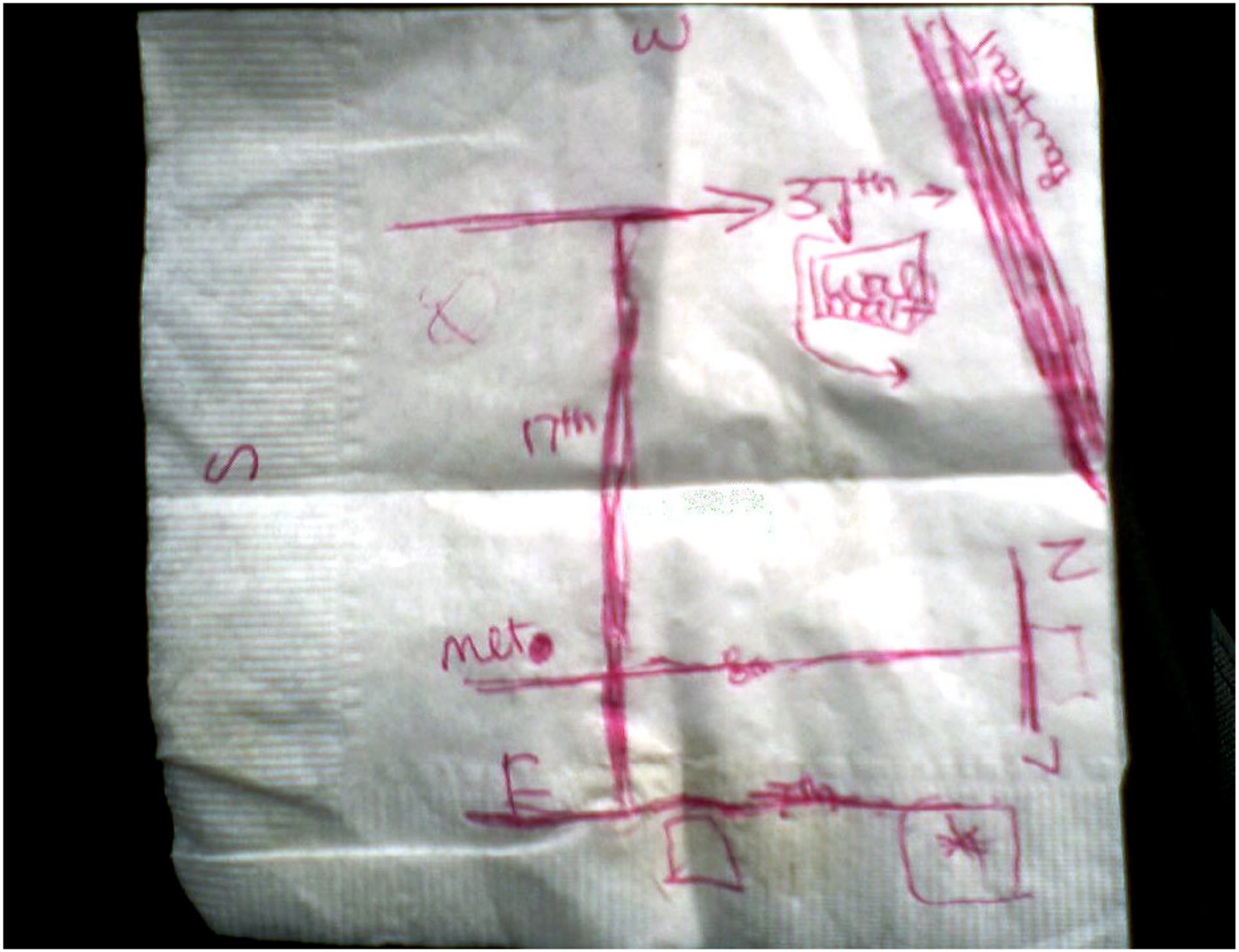
- but you gotta love software models…I do

Why build models?

- Modelling can guide your exploration:
  - It can help you figure out what questions to ask
  - It can help to reveal key design decisions
  - It can help you to uncover problems
- Modelling can help us check our understanding
  - Reason about the model to understand its consequences
    - Does it have the properties we expect?
  - Animate the model to help us visualize/validate software behaviour
- Modelling can help us communicate
  - Provides useful abstractions that focus on the point you want to make…
  - …without overwhelming people with detail
- Throw-away modelling?
  - The exercise of modelling is more important than the model itself
  - Time spent perfecting the models might be time wasted…

Maps as Abstractions




W
37th
freeway
17th
S
met
8th
N
E
7th

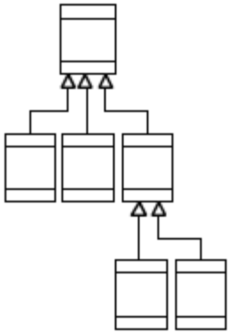
Dealing with problem complexity

- **Abstraction**
  - Ignore detail to see the big picture
  - Treat objects as the same by ignoring certain differences
  - (beware: every abstraction involves choice over what is important)
- **Decomposition**
  - Partition a problem into independent pieces, to study separately
  - (beware: the parts are rarely independent really)
- **Projection**
  - Separate different concerns (views) and describe them separately
  - Different from decomposition as it does not partition the problem space
  - (beware: different views will be inconsistent most of the time)
- **Modularization**
  - Choose structures that are stable over time, to localize change
  - (beware: any structure will make some changes easier and others harder)

the Unified Modelling Language (UML)

- → Third generation OO method
  - ➥ Booch, Rumbaugh & Jacobson are principal authors
    - ➢ Still evolving (currently version 2.0)
    - ➢ Attempt to standardize the proliferation of OO variants
  - ➥ Is purely a notation
    - ➢ No modelling method associated with it!
    - ➢ Was intended as a design notation
  - ➥ Has become an industry standard
    - ➢ But is primarily promoted by IBM/Rational (who sell lots of UML tools, services)
- → Has a standardized meta-model
  - ➥ Use case diagrams
  - ➥ Class diagrams
  - ➥ Message sequence charts
  - ➥ Activity diagrams
  - ➥ State Diagrams
  - ➥ Module Diagrams
  - ➥ Platform diagrams
  - ➥ …

Modeling Notations

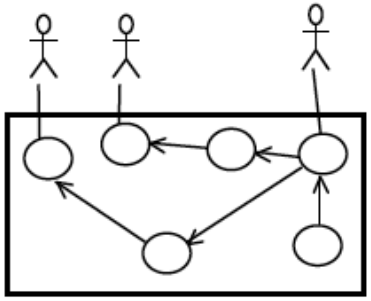


UML Class Diagrams

information structure

relationships between data items

modular structure for the system

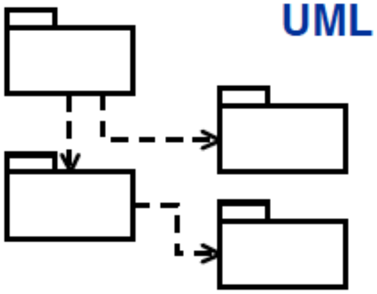


Use Cases

user's view

Lists functions

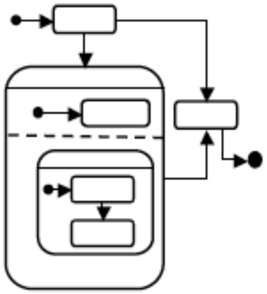
visual overview of the main requirements



UML Package Diagrams

Overall architecture

Dependencies between components

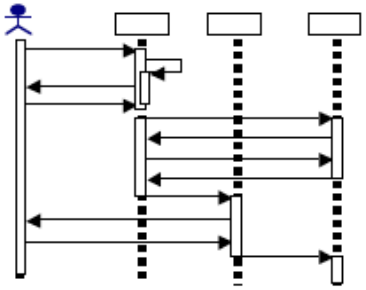


(UML) Statecharts

responses to events

dynamic behavior

event ordering, reachability, deadlock, etc

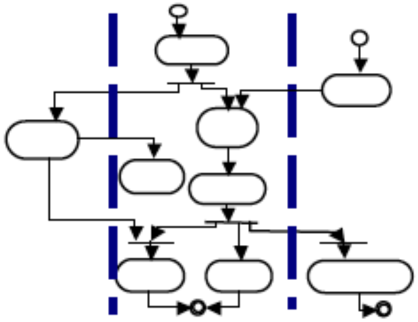


UML Sequence Diagrams

individual scenario

interactions between users and system

Sequence of messages



Activity diagrams

business processes;

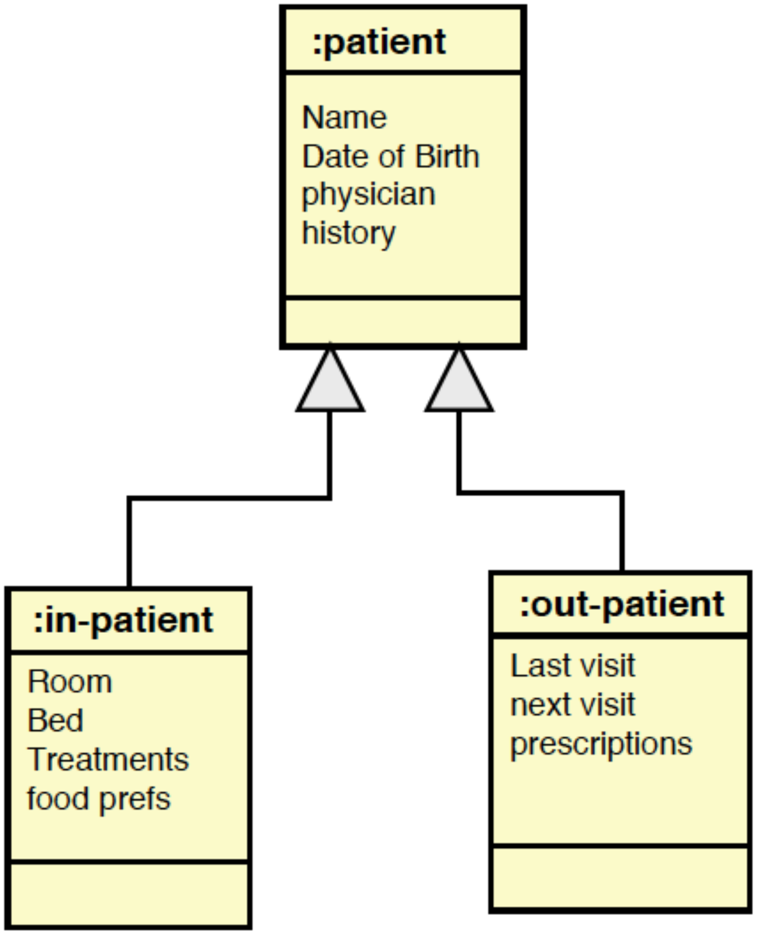
concurrency and synchronization;

dependencies between tasks;

Intro: Object Classes in UML

Source: *Adapted from Davis, 1990, p67-68*

Generalization
(an abstraction hierarchy)

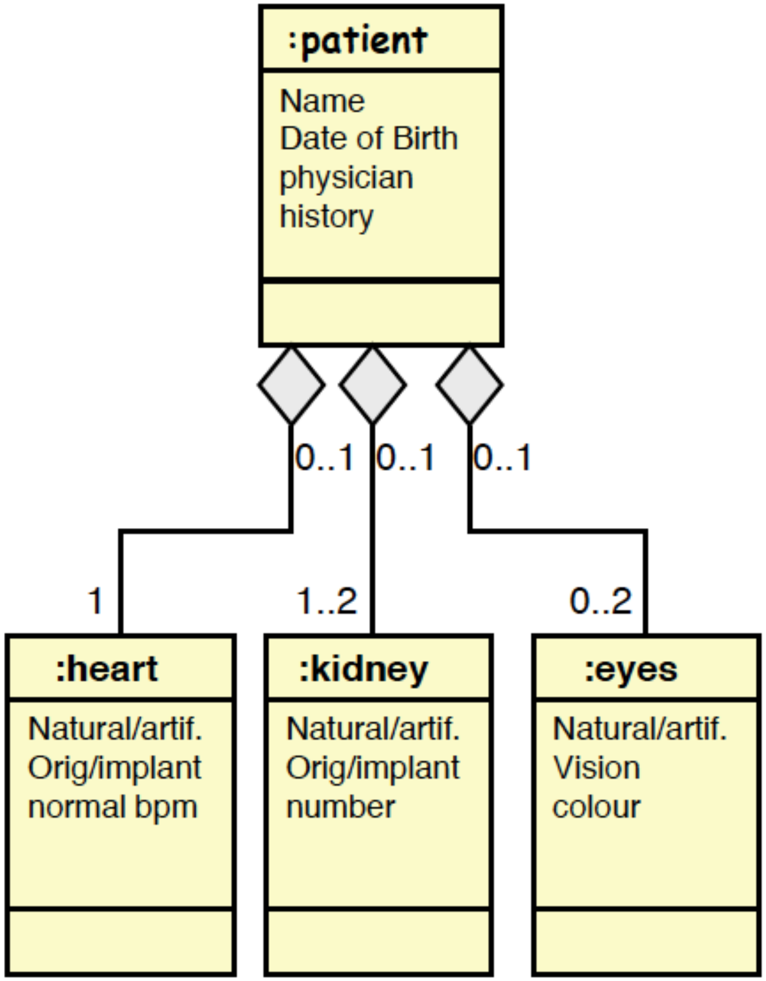


:patient
Name
Date of Birth
physician
history

:in-patient
Room
Bed
Treatments
food prefs

:out-patient
Last visit
next visit
prescriptions

Aggregation
(a partitioning hierarchy)

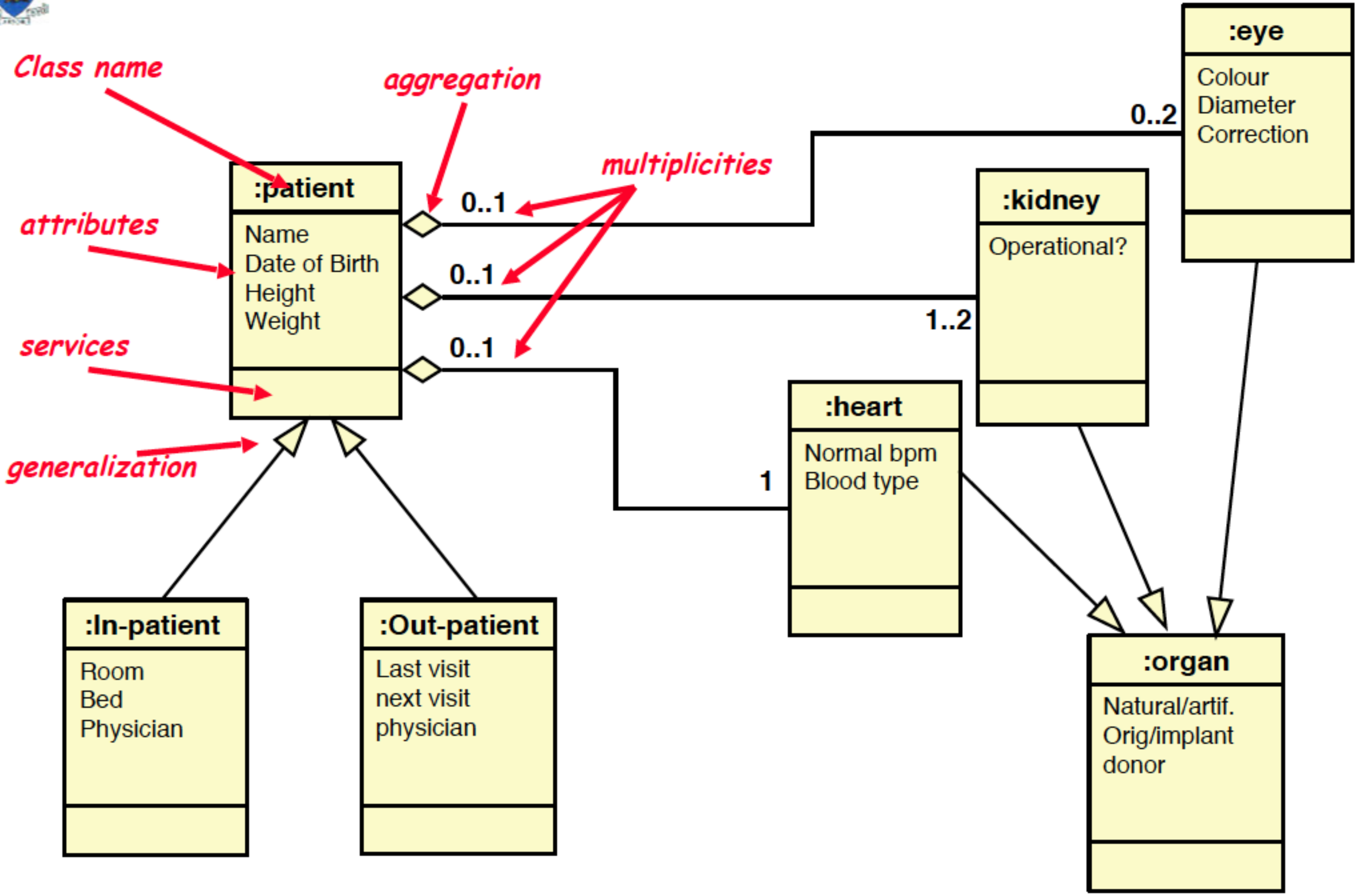


:patient
Name
Date of Birth
physician
history

:heart
Natural/artif.
Orig/implant
normal bpm

:kidney
Natural/artif.
Orig/implant
number

:eyes
Natural/artif.
Vision
colour



Class name

aggregation

multiplicities

attributes

services

generalization

:patient

Name
Date of Birth
Height
Weight

0..1

0..1

0..1

:eye

Colour
Diameter
Correction

0..2

:kidney

Operational?

1..2

:heart

Normal bpm
Blood type

1

:In-patient

Room
Bed
Physician

:Out-patient

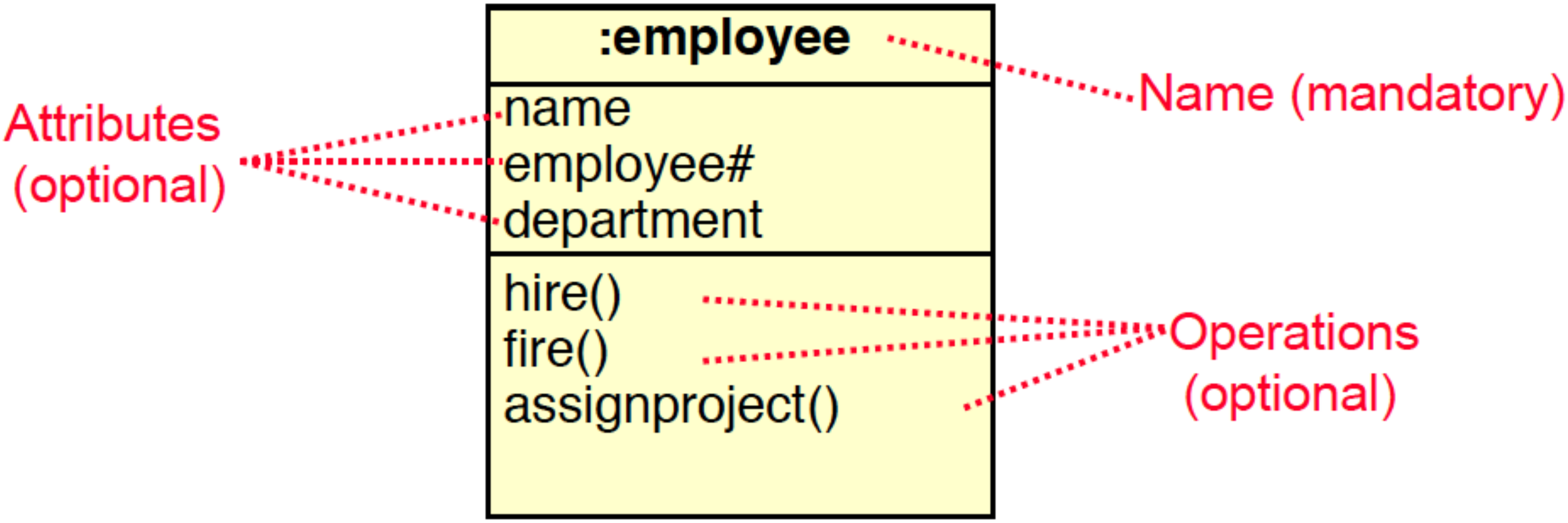
Last visit
next visit
physician

:organ

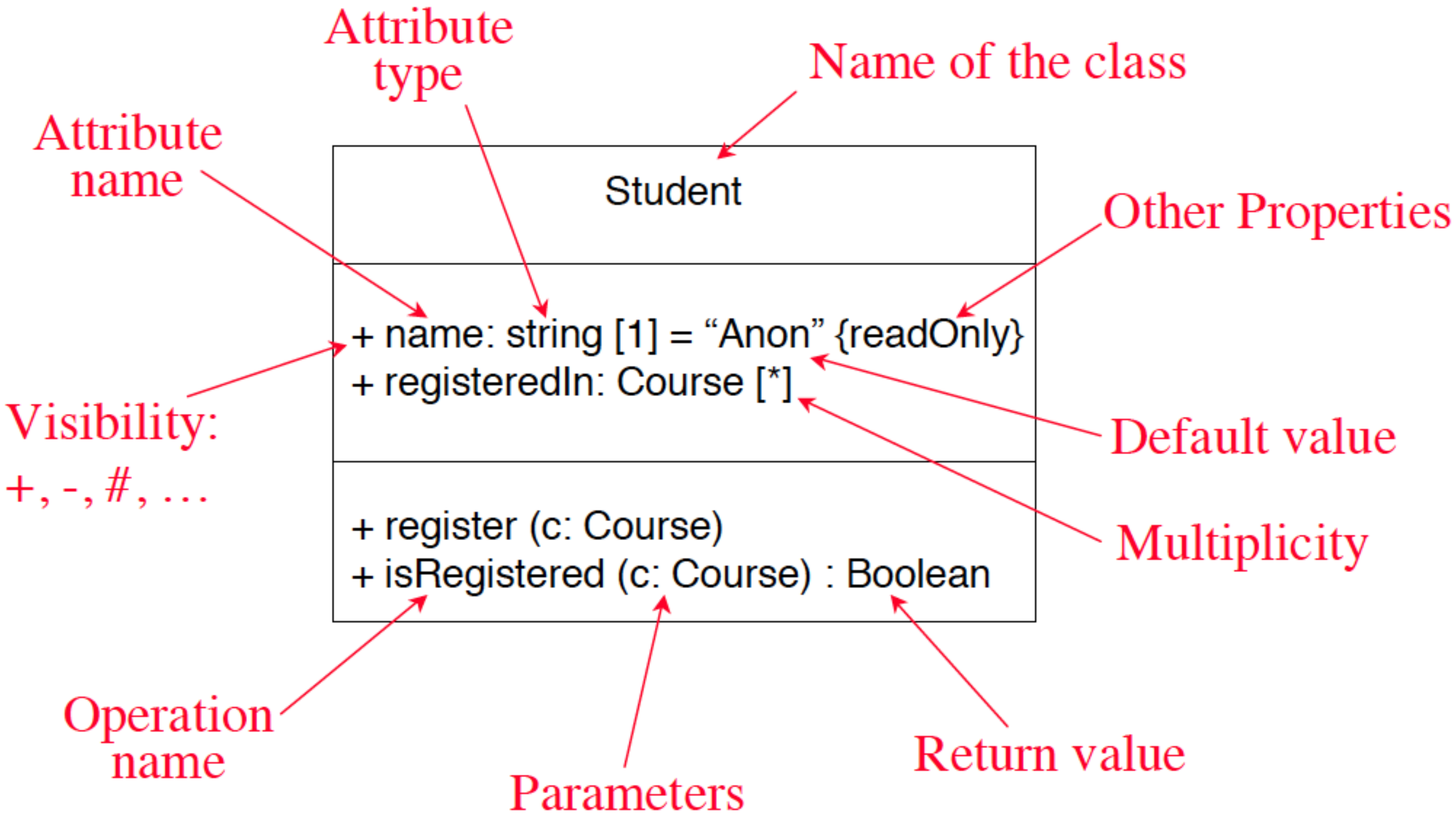
Natural/artif.
Orig/implant
donor

What are classes?

- A class describes a group of objects with
  - similar properties (attributes),
  - common behaviour (operations),
  - common relationships to other objects,
  - and common meaning ("semantics").
- Examples
  - employee: has a name, employee# and department; an employee is hired, and fired; an employee works in one or more projects

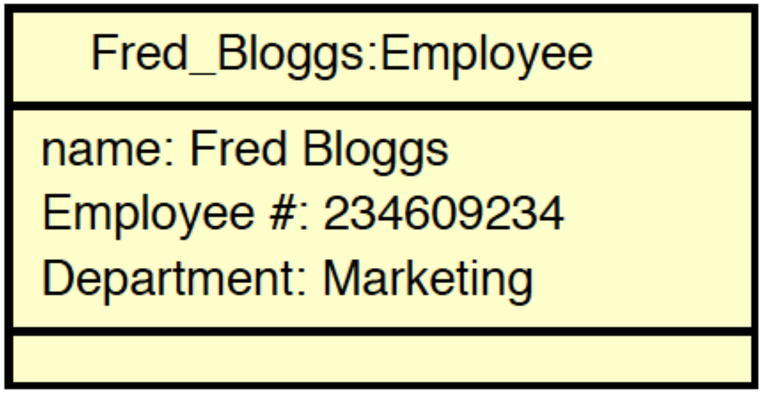


The full notation…



Objects vs. Classes

- The instances of a class are called objects.
  - Objects are represented as:



Fred_Bloggs:Employee

name: Fred Bloggs

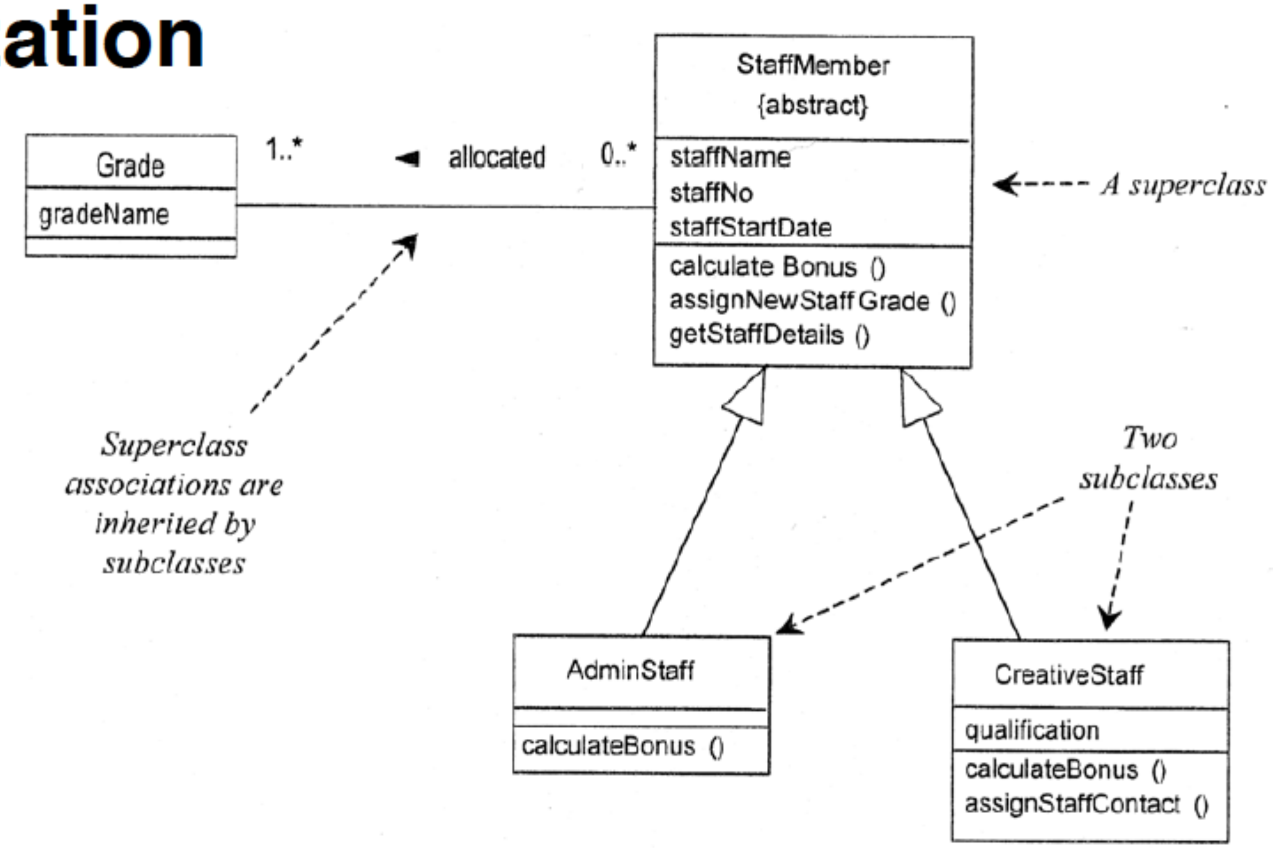
Employee #: 234609234

Department: Marketing

  - Two different objects may have identical attribute values (like two people with identical name and address)

- Objects have associations with other objects
  - E.g. Fred_Bloggs:employee is associated with the KillerApp:project object
  - But we will capture these relationships at the class level (why?)
  - Note: Make sure attributes are associated with the right class
    - E.g. you don’t want both managerName and manager# as attributes of Project! (…Why??)

Generalization



→ Notes:

- Subclasses inherit attributes, associations, & operations from the superclass
- A subclass may override an inherited aspect
  - e.g. AdminStaff & CreativeStaff have different methods for calculating bonuses
- Superclasses may be declared {abstract}, meaning they have no instances
  - Implies that the subclasses cover all possibilities
  - e.g. there are no other staff than AdminStaff and CreativeStaff

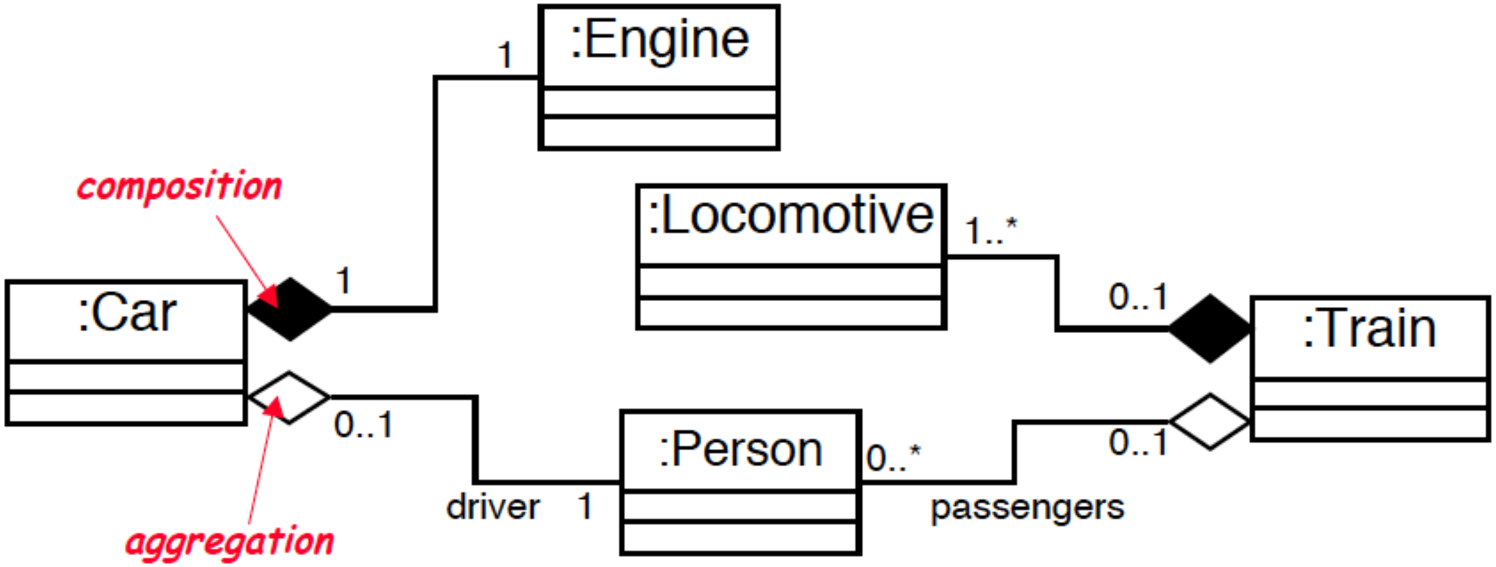
Aggregation and Composition

→ Aggregation

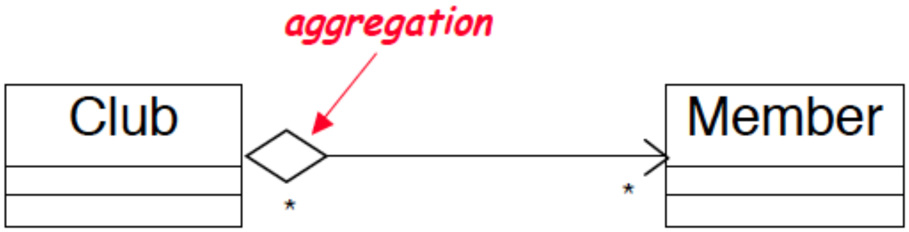
- This is the "Has-a" or "Whole/part" relationship

→ Composition

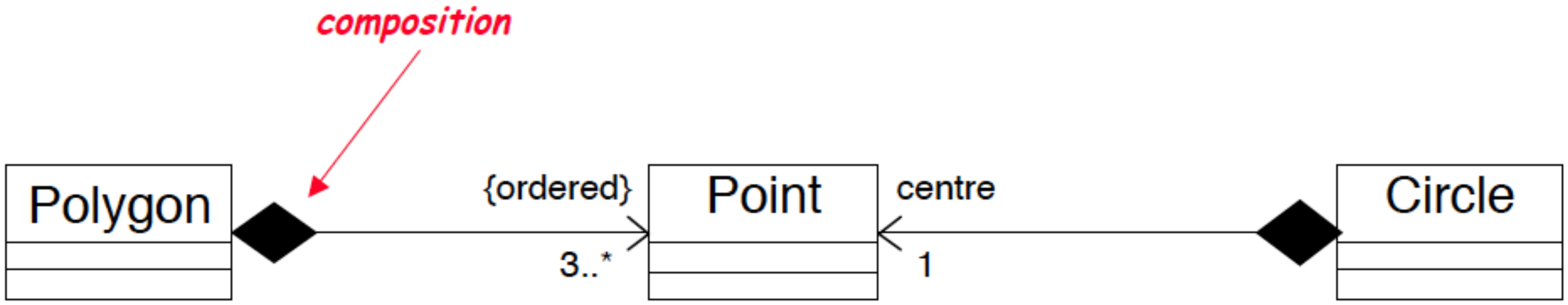
- Strong form of aggregation that implies ownership:
  - if the whole is removed from the model, so is the part.
  - the whole is responsible for the disposition of its parts



Aggregation / Composition (Refresher)



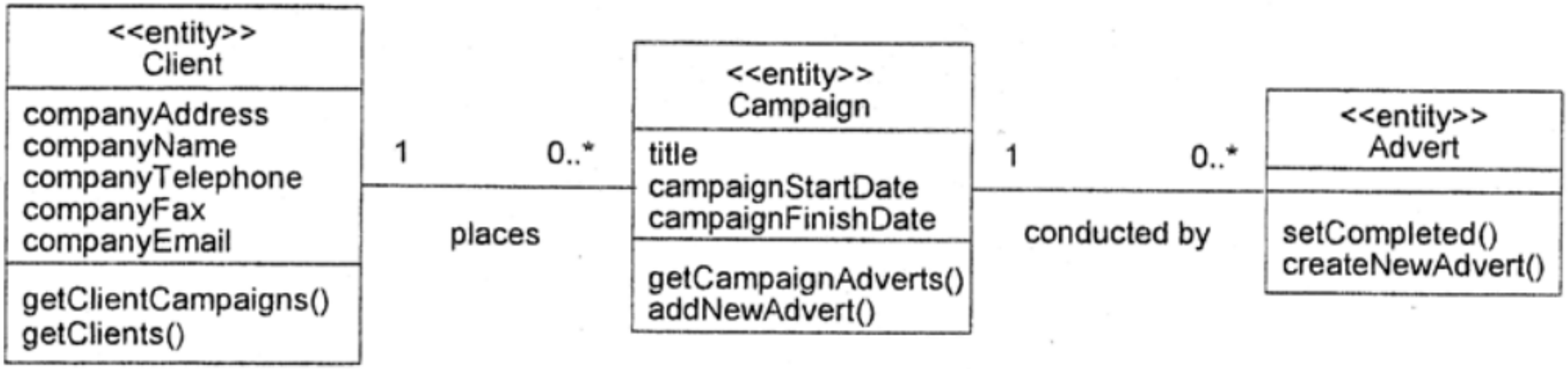
What does this mean??



Note: No sharing - any instance of point can be part of a polygon or a circle, but not both

Associations

- → Objects do not exist in isolation from one another
  - A relationship represents a connection among things.
  - In UML, there are different types of relationships:
    - Association
    - Aggregation and Composition
    - Generalization
    - Dependency
    - Realization
- → Class diagrams show classes and their relationships



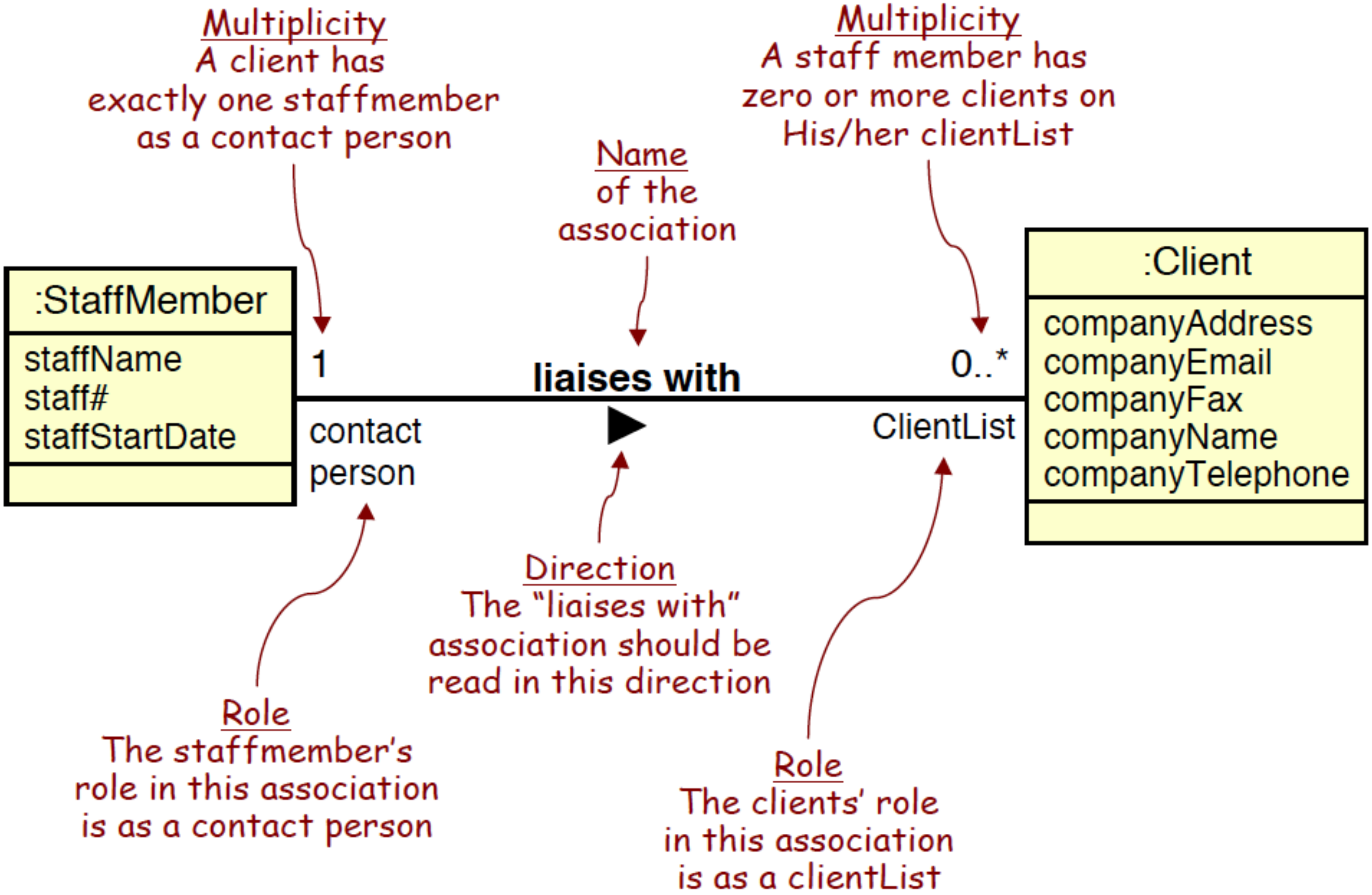
Association Multiplicity

- Ask questions about the associations:
  - Can a campaign exist without a member of staff to manage it?
    - If yes, then the association is optional at the Staff end - zero or more (0..*)
    - If no, then it is not optional - one or more (1..*)
    - If it must be managed by one and only one member of staff - exactly one (1)
  - What about the other end of the association?
    - Does every member of staff have to manage exactly one campaign?
    - No. So the correct multiplicity is zero or more.

- Some examples of specifying multiplicity:

Optional (0 or 1)	0..1	
Exactly one	1	= 1..1
Zero or more	0..*	= *
One or more	1..*	
A range of values	2..6	

Class associations



Multiplicity
A client has exactly one staffmember as a contact person

Name of the association

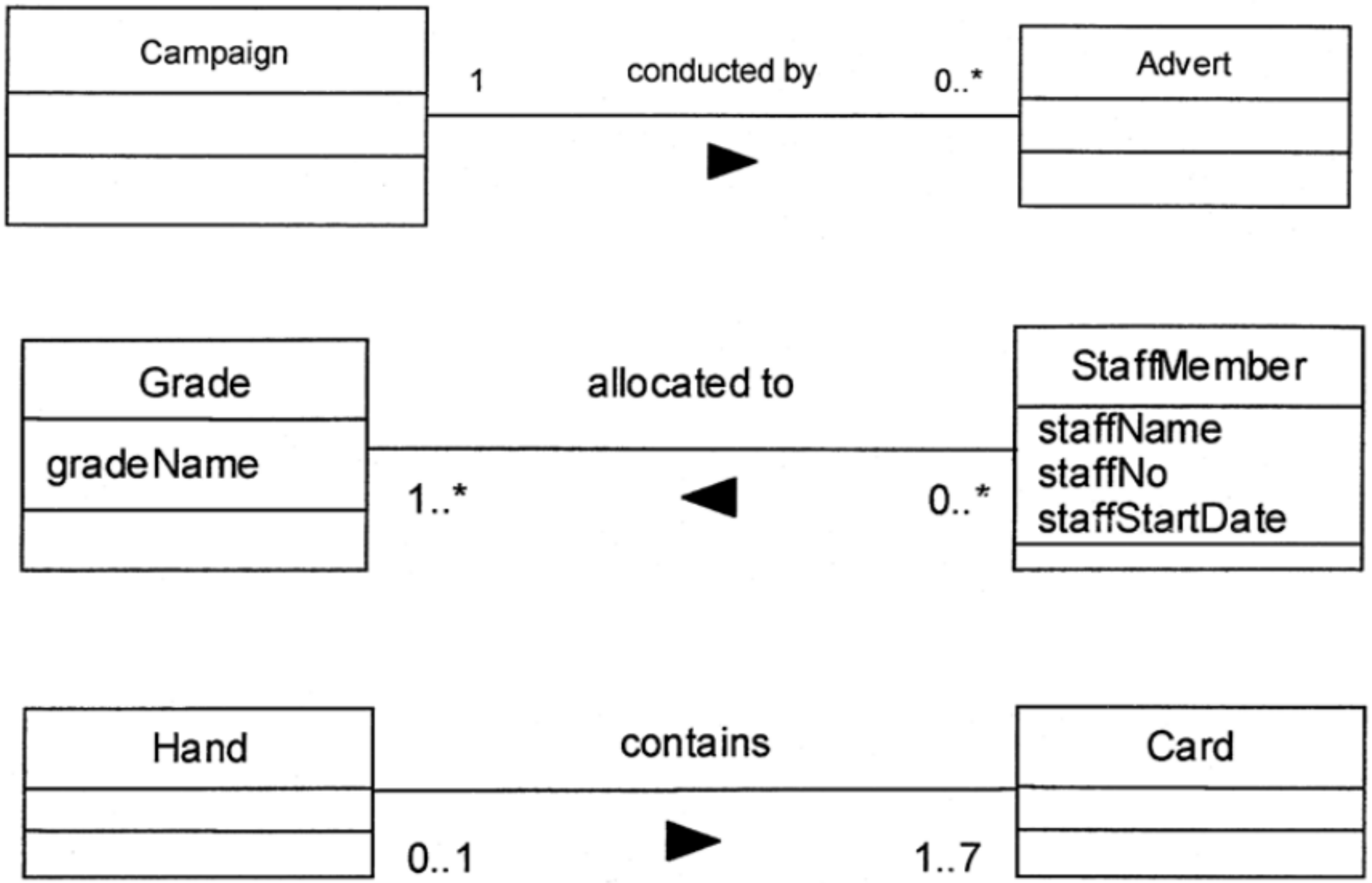
Multiplicity
A staff member has zero or more clients on His/her clientList

Role
The staffmember's role in this association is as a contact person

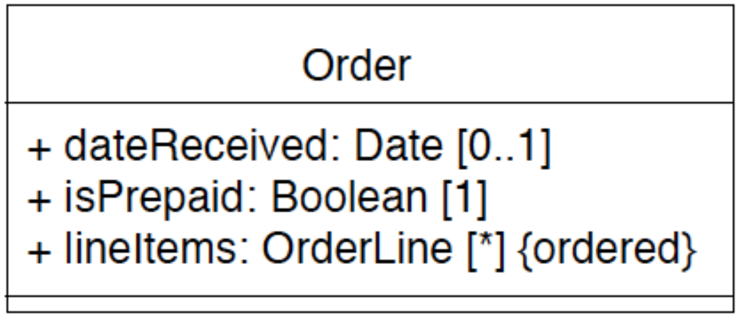
Direction
The "liaises with" association should be read in this direction

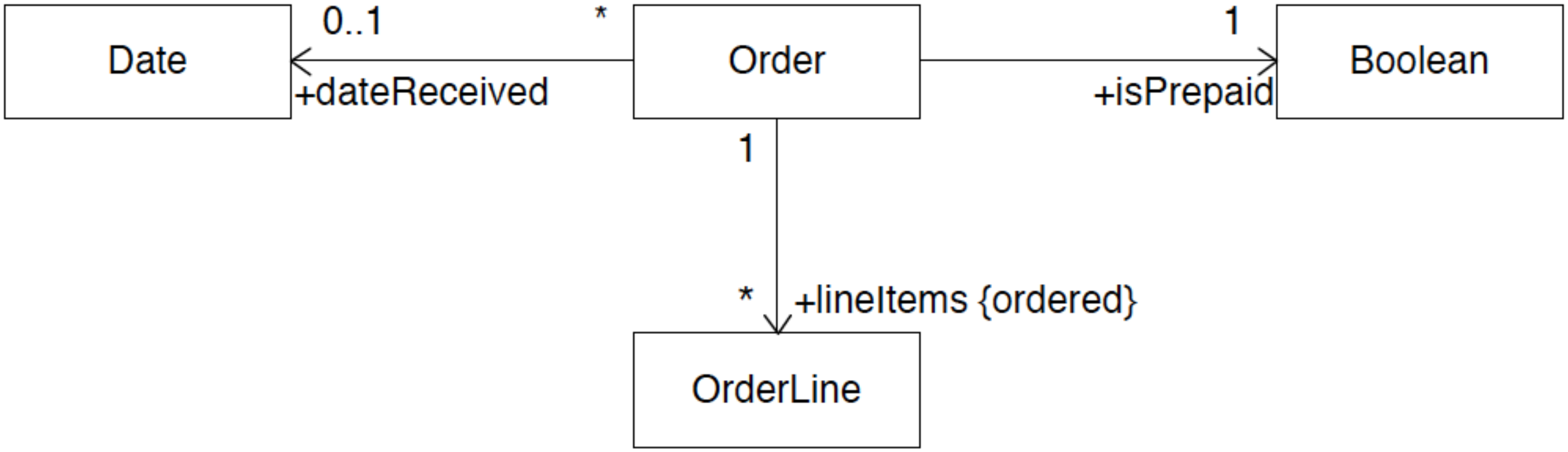
Role
The clients' role in this association is as a clientList

More Examples

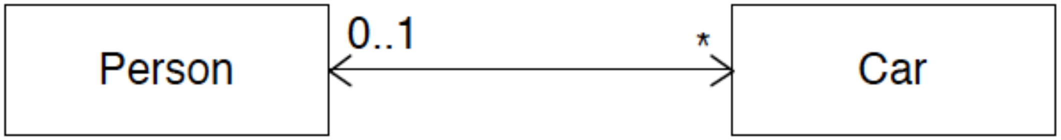


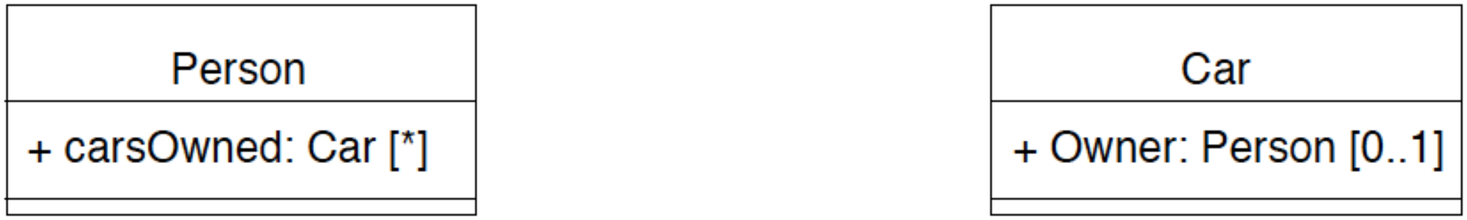
Navigability / Visibility


Order
+ dateReceived: Date [0..1]
+ isPrepaid: Boolean [1]
+ lineItems: OrderLine [*] {ordered}



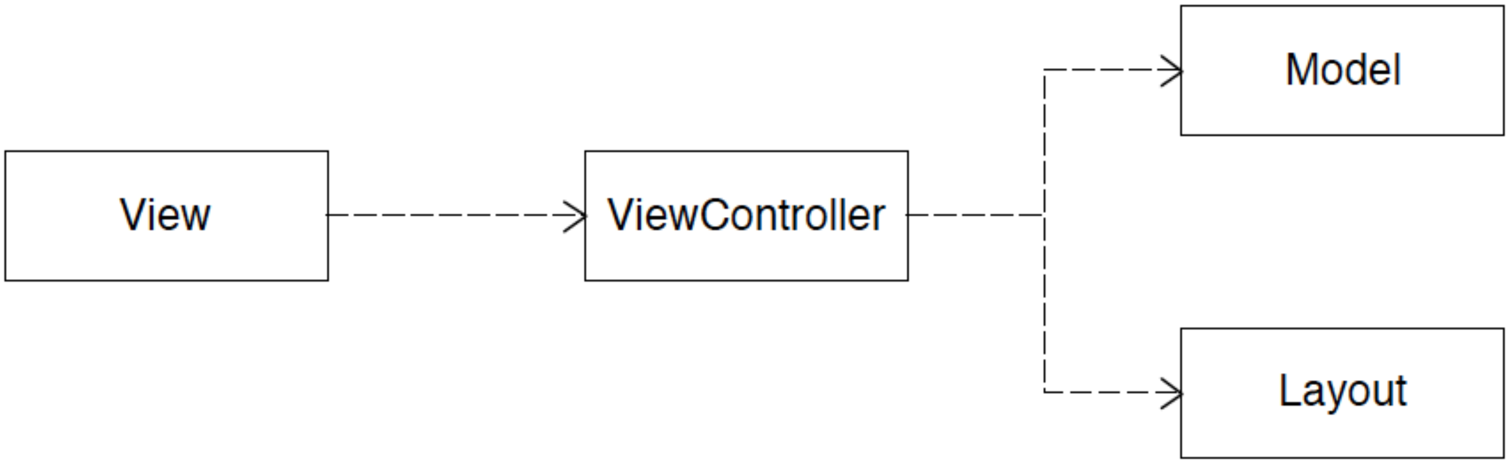
Bidirectional Associations





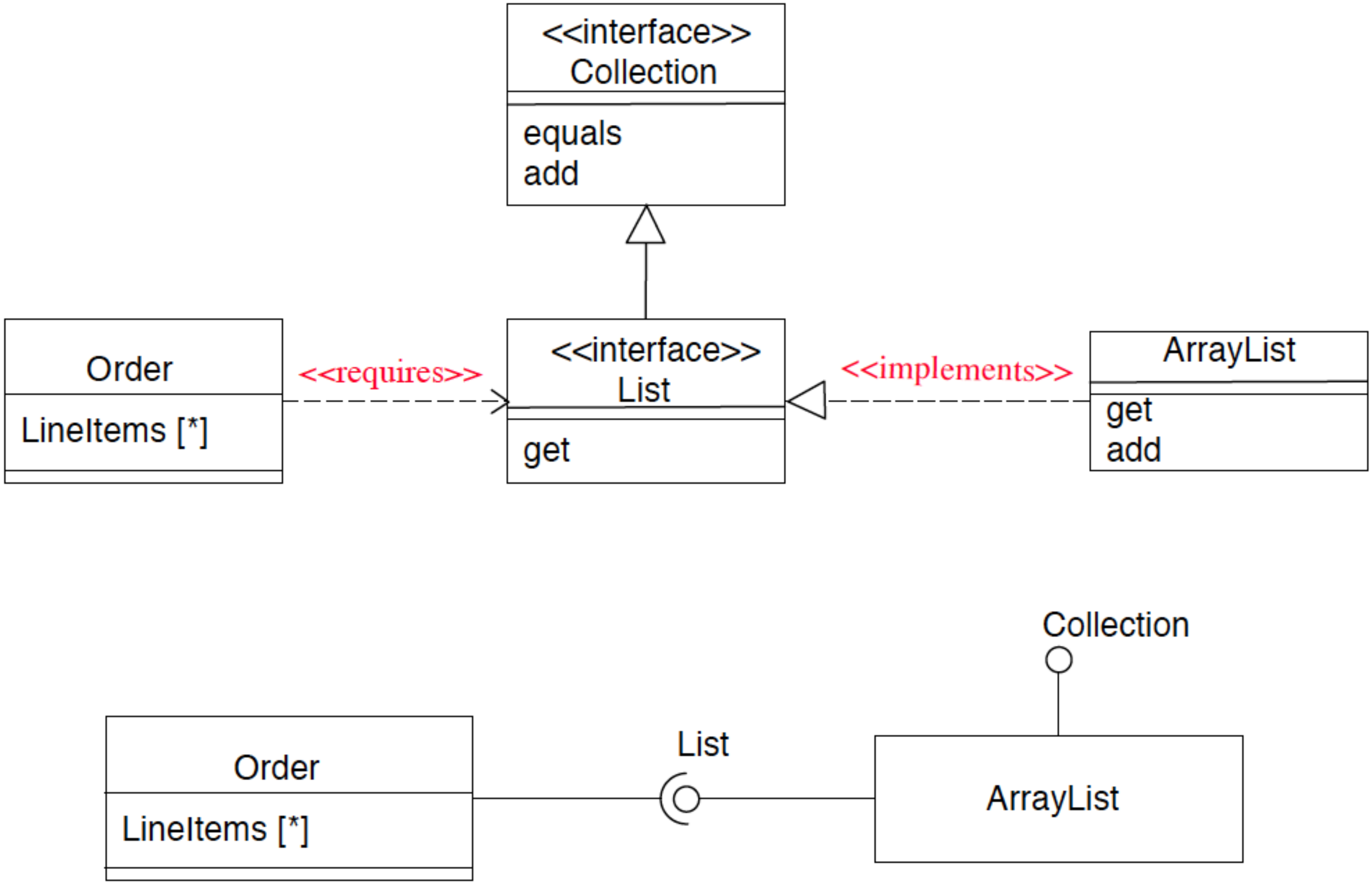
Hard to implement correctly!

Dependencies



- **Example Dependency types:**
  - `<<call>>`
  - `<<use>>`
  - `<<create>>`
  - `<<derive>>`
  - `<<instantiate>>`
  - `<<permit>>`
  - `<<realize>>`
  - `<<refine>>`
  - `<<substitute>>`
  - `<<parameter>>`

Interfaces

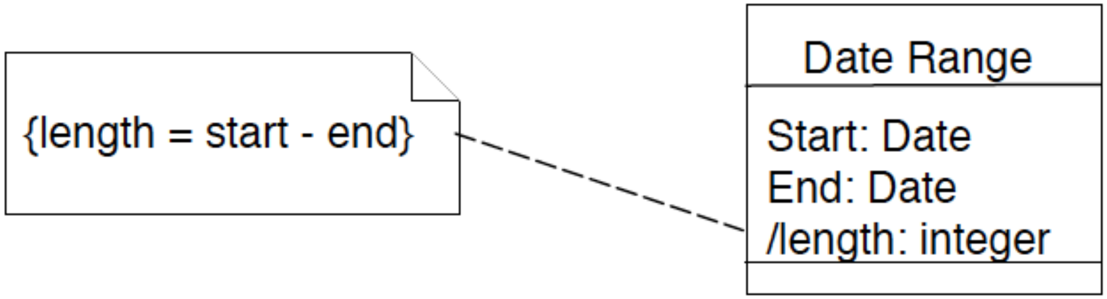

<<interface>>
Collection
equals
add
Order
LineItems [*]
<<requires>>
<<interface>>
List
get
<<implements>>
ArrayList
get
add
Collection
Order
LineItems [*]
List
ArrayList

Annotations

→ **Comments**

- -- can be used to add comments within a class description

→ **Notes**



→ **Constraint Rules**

- Any further constraints {in curly braces}
- e.g. {time limit: length must not be more than three months}

What UML class diagrams can show

- Division of Responsibility
  - Operations that objects are responsible for providing
- Subclassing
  - Inheritance, generalization
- Navigability / Visibility
  - When objects need to know about other objects to call their operations
- Aggregation / Composition
  - When objects are part of other objects
- Dependencies
  - When changing the design of a class will affect other classes
- Interfaces
  - Used to reduce coupling between objects

static vs. dynamic modeling

- static captures fixed code-level relationships
  - class (and package) diagrams
  - object diagrams
  - component diagrams
  - deployment diagrams
- behavioral diagrams capture dynamic execution
  - use case diagrams
  - sequence and interaction diagrams
  - collaboration diagrams
  - statechart diagrams
  - activity diagrams

summary

- summary on modeling
  - important to use modeling during design
  - modeling can be helpful to discover design and architecture (a1)
  - as with most things, it can be taken too far
  - the model should provide an easier to consume abstraction
  - strict uml is good when publishing designs for external consumption even if you don’t use it yourself