CSC C01
Introduction to Modelling & UML

Anya Tafliovich

with thanks to S. Easterbrook

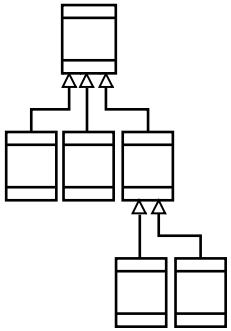
Why build models?

- **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…

the Unified Modelling Language (UML)

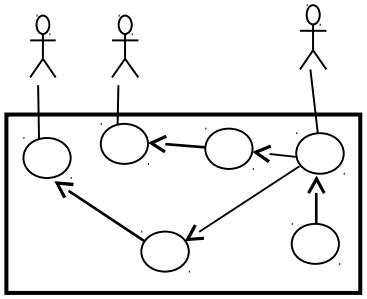
- **Third generation OO method**
  - Booch, Rumbaugh & Jacobson are principal authors
    - Still evolving (currently version 2.0)
    - Attempt to standardize the proliferation of OO variants
  - Is purely a notation
    - No modelling method associated with it!
    - Was intended as a design notation
  - 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
  - Activity diagrams
  - State Diagrams
  - Module Diagrams
  - Platform diagrams
  - ...

Modelling Notations



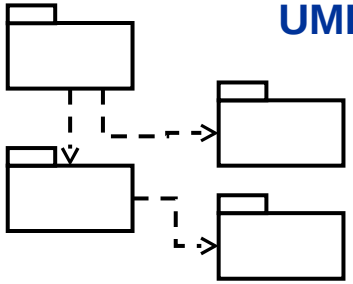
UML Class Diagrams

- information structure
- relationships between data items
- modular structure for the system



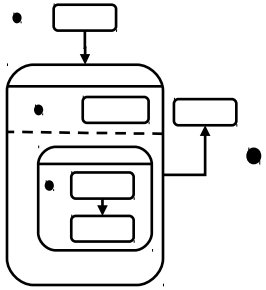
Use Cases

- user's view
- Lists functions
- visual overview of the main requirements



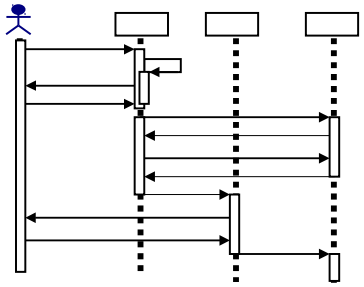
UML Package Diagrams

- Overall architecture
- Dependencies between components



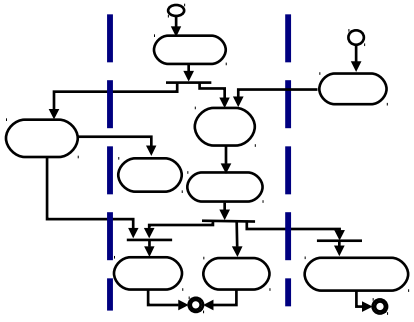
(UML) Statecharts

- responses to events
- dynamic behavior
- event ordering, reachability, deadlock, etc



UML Sequence Diagrams

- individual scenario
- interactions between users and system
- Sequence of messages



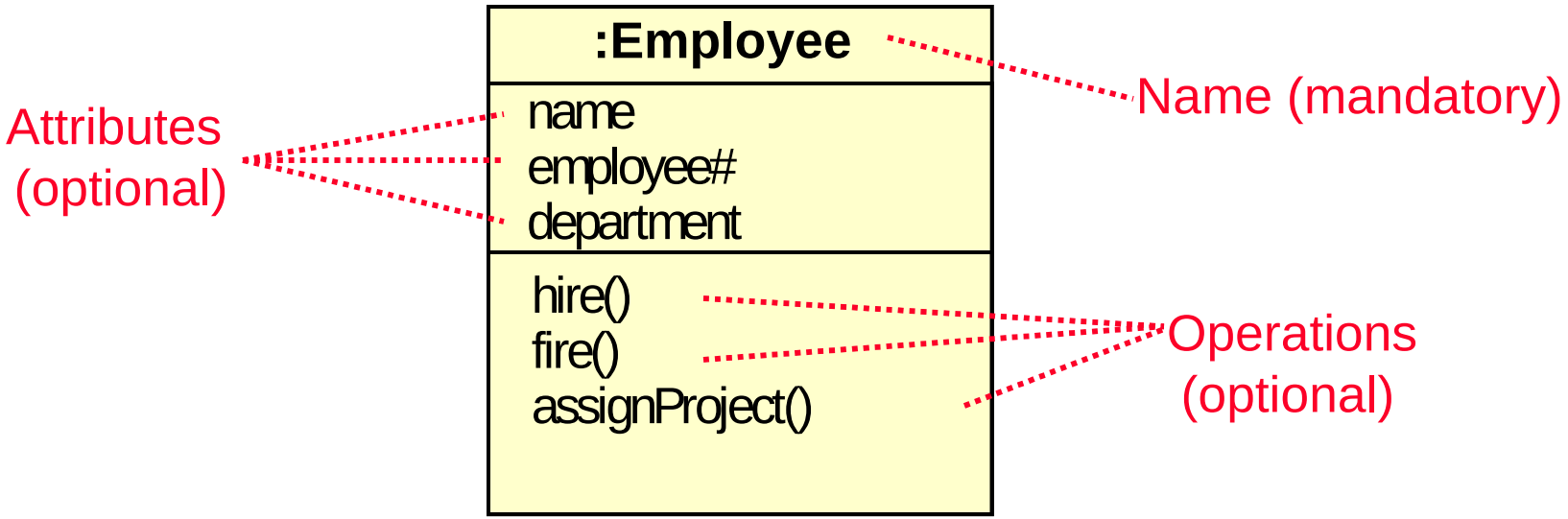
Activity diagrams

- business processes;
- concurrency and synchronization;
- dependencies between tasks;

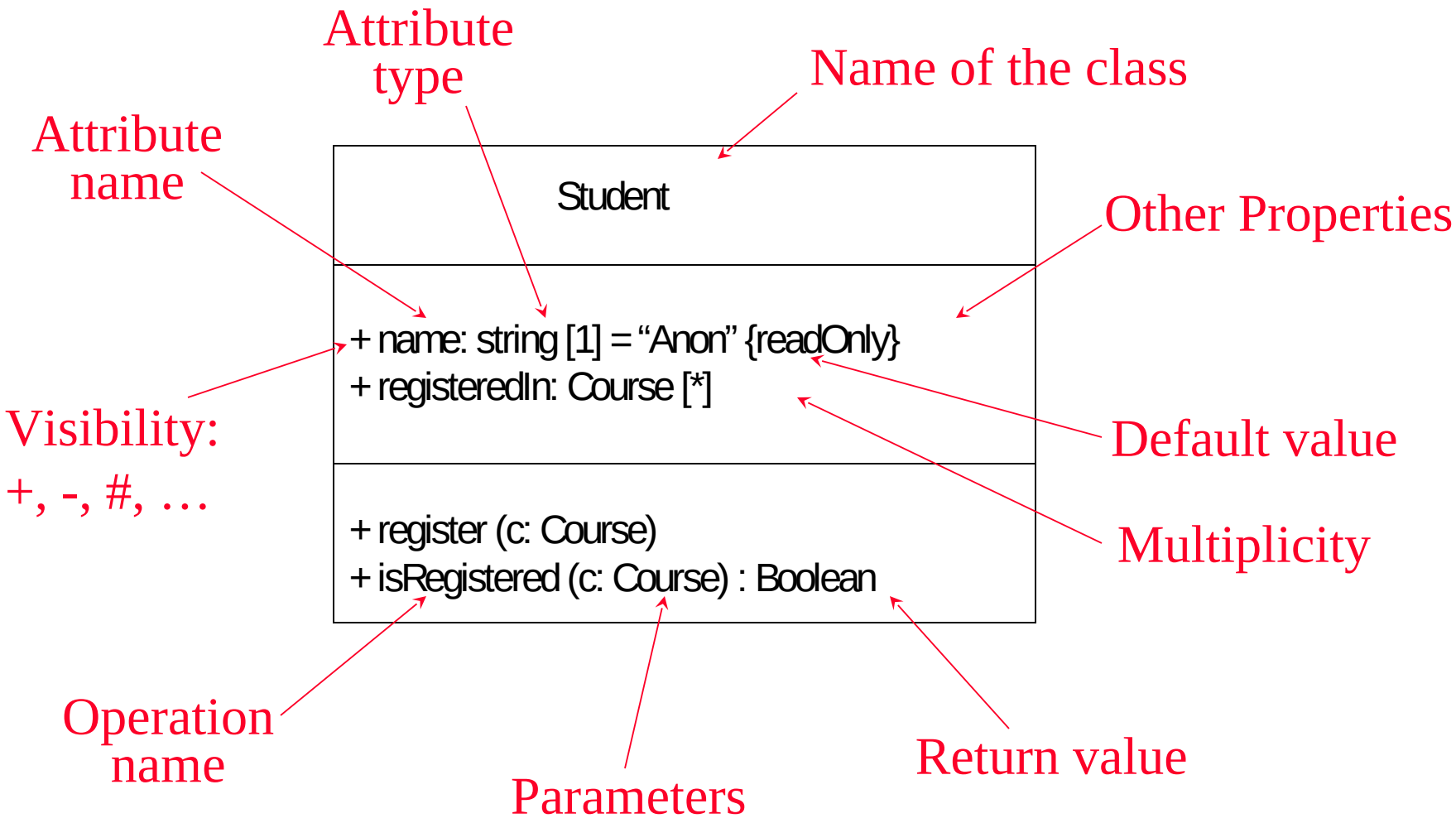
What are classes?

- **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**

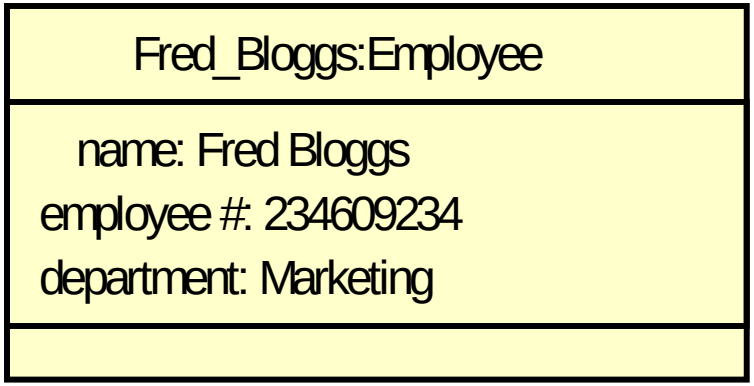


The full notation…



Objects vs. Classes

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

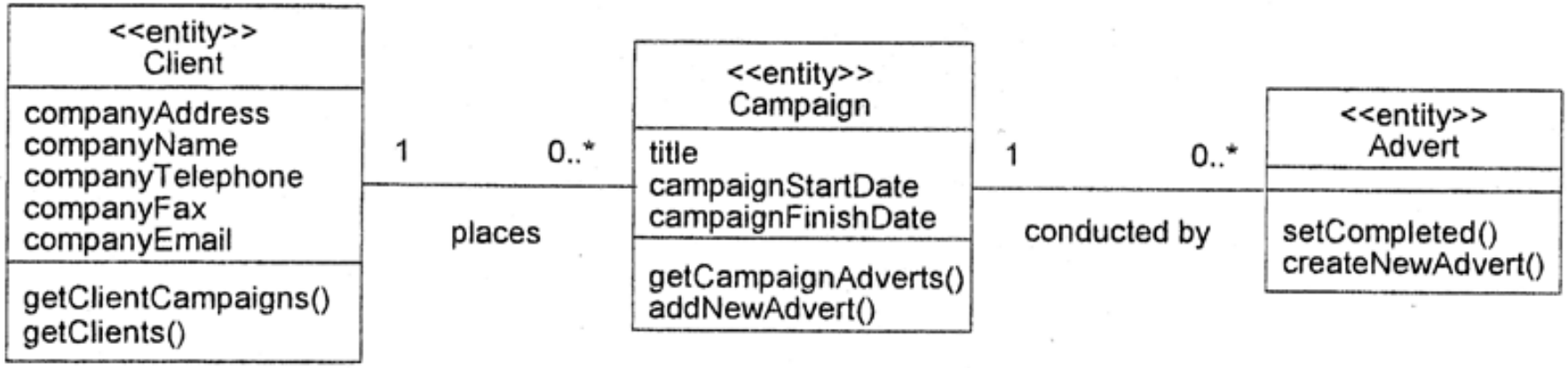


  - **Two different objects may have identical attribute values (like two people with identical name and address)**

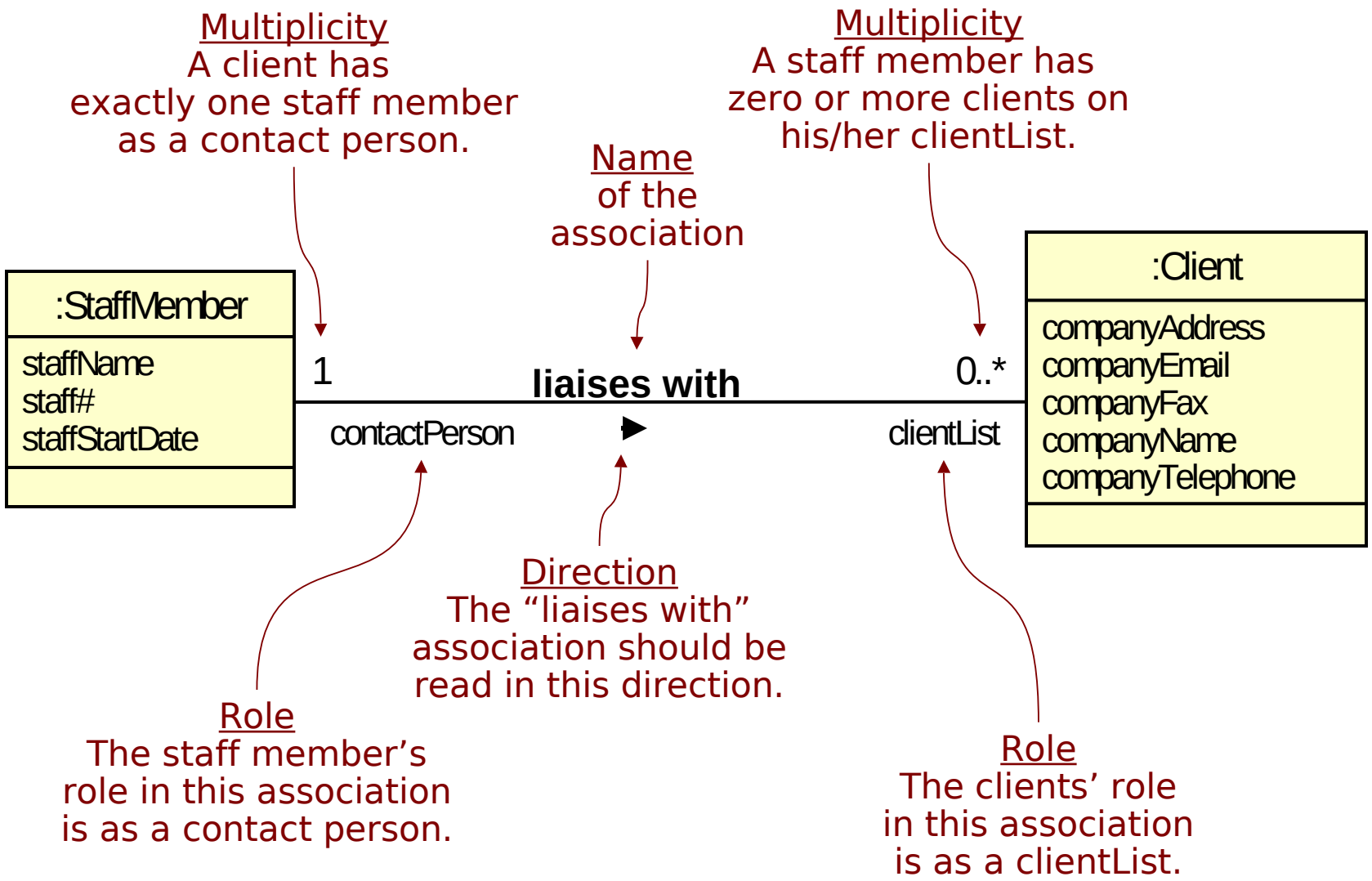
- **Objects have associations with other objects**
  - **E.g., Fred_Bloggs:Employee is associated with the KillerApp:Project object**
  - **But we will capture these relationships at the class level (why?)**
  - **Note: Make sure attributes are associated with the right class**
    - **E.g. you don't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

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**



Class associations

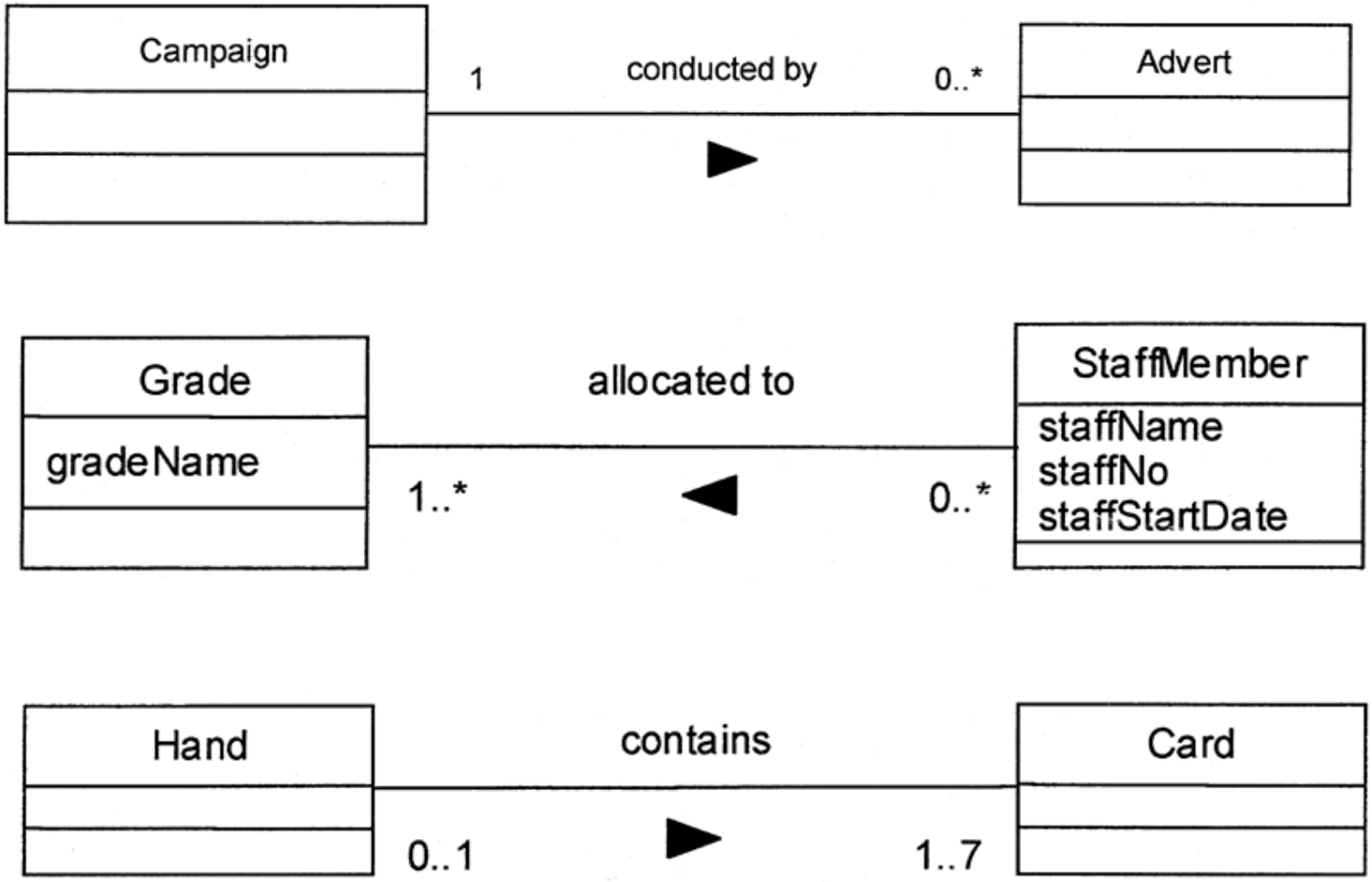


Association Multiplicity

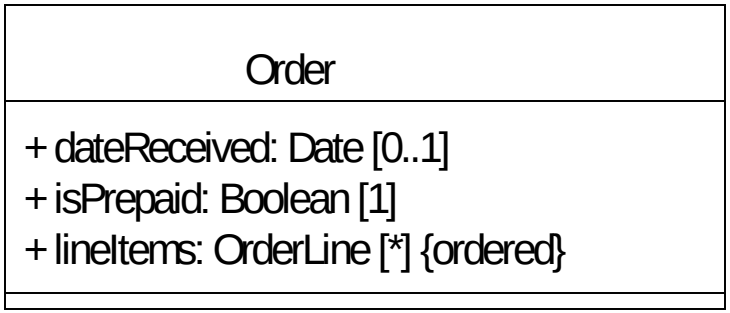
- **Ask questions about the associations:**
  - Can a campaign exist without a member of staff to manage it?
    - If yes, then the association is optional at the Staff end - zero or more (0..*)
    - If no, then it is not optional - one or more (1..*)
    - If it must be managed by one and only one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

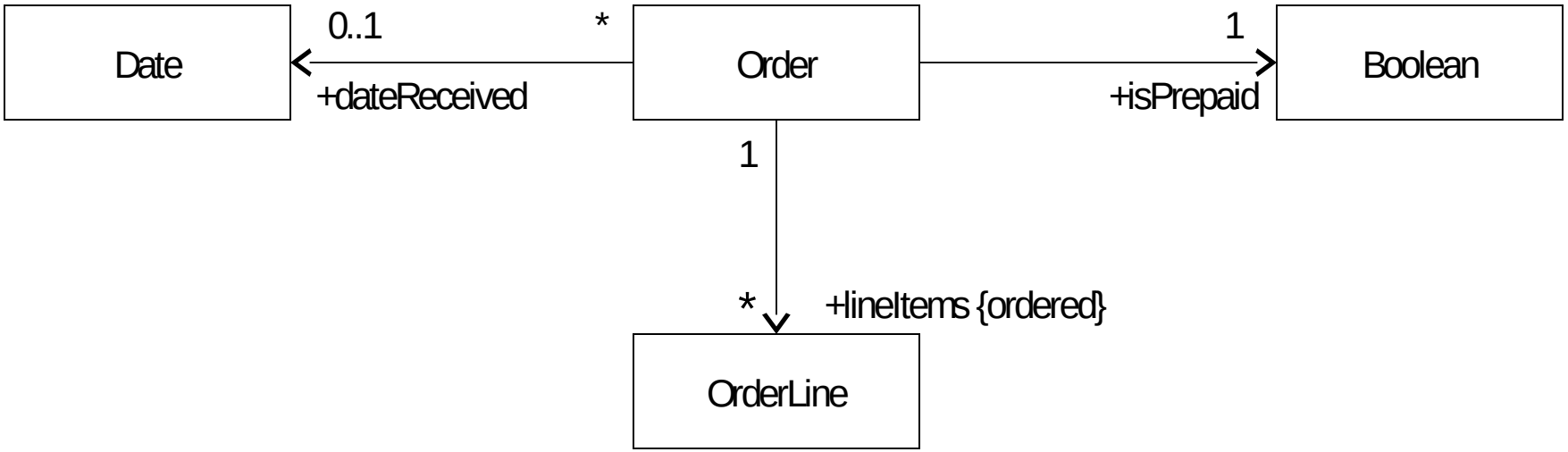
- **Some examples of specifying multiplicity:**
  - Optional (0 or 1) 0..1
  - Exactly one 1 = 1..1
  - Zero or more 0..* = *
  - One or more 1..*
  - A range of values 2..6

More Examples

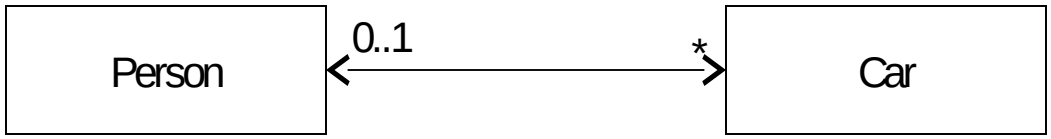


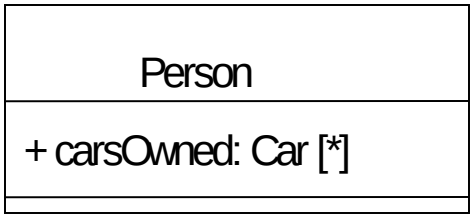
Navigability / Visibility

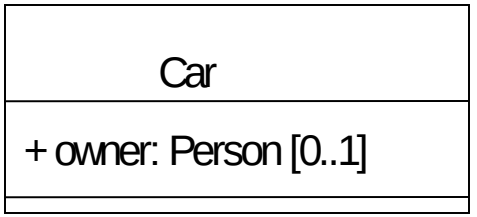




Bidirectional Associations





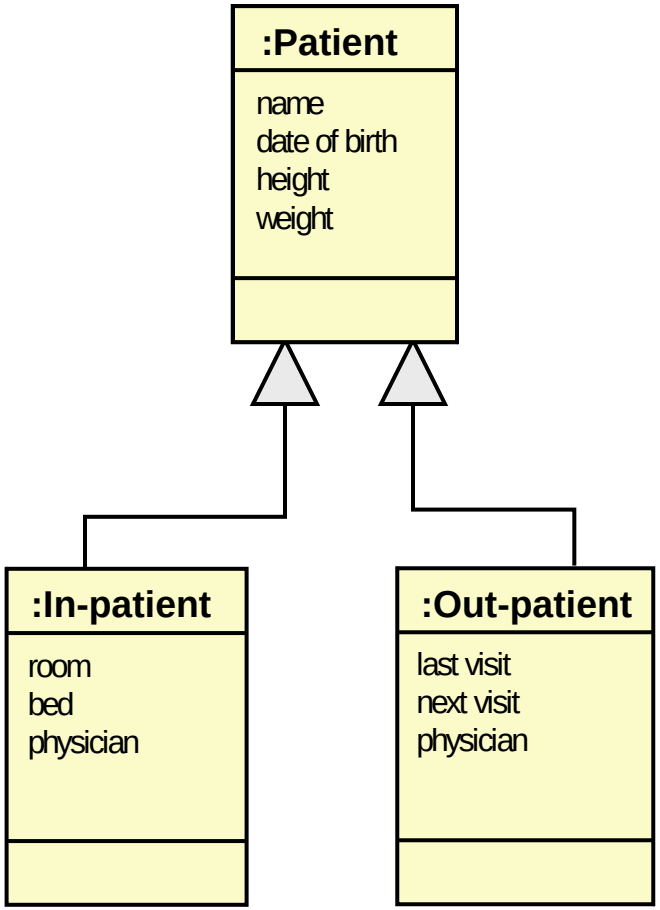


Hard to implement correctly!

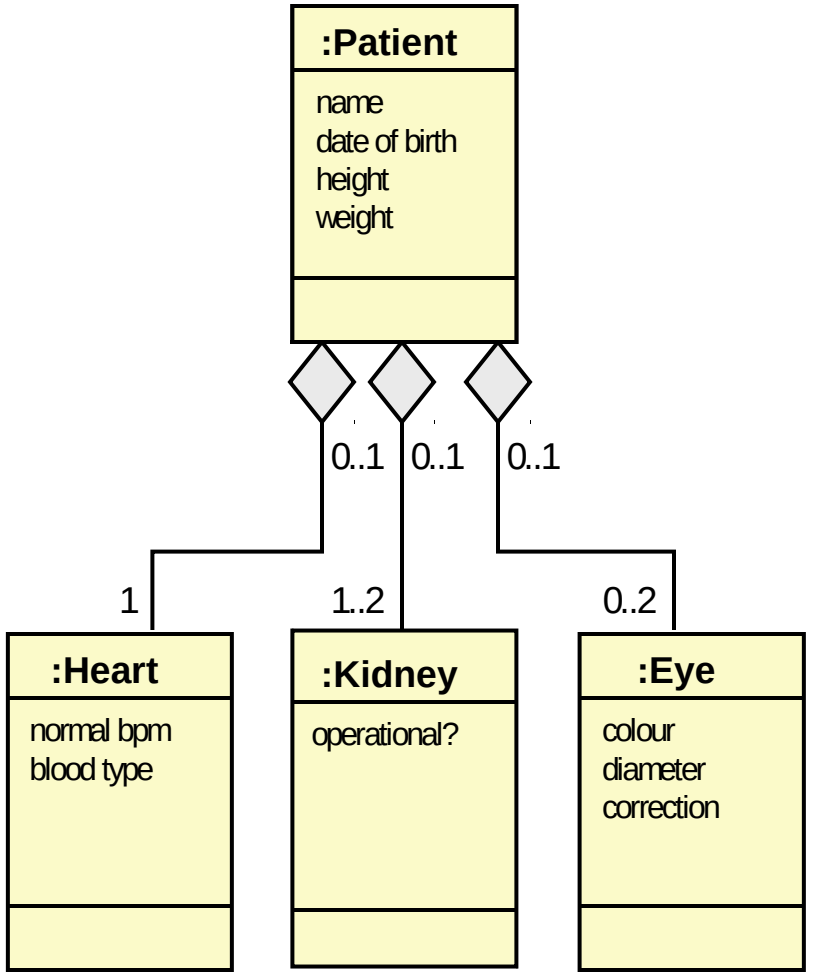
Generalization and Aggregation

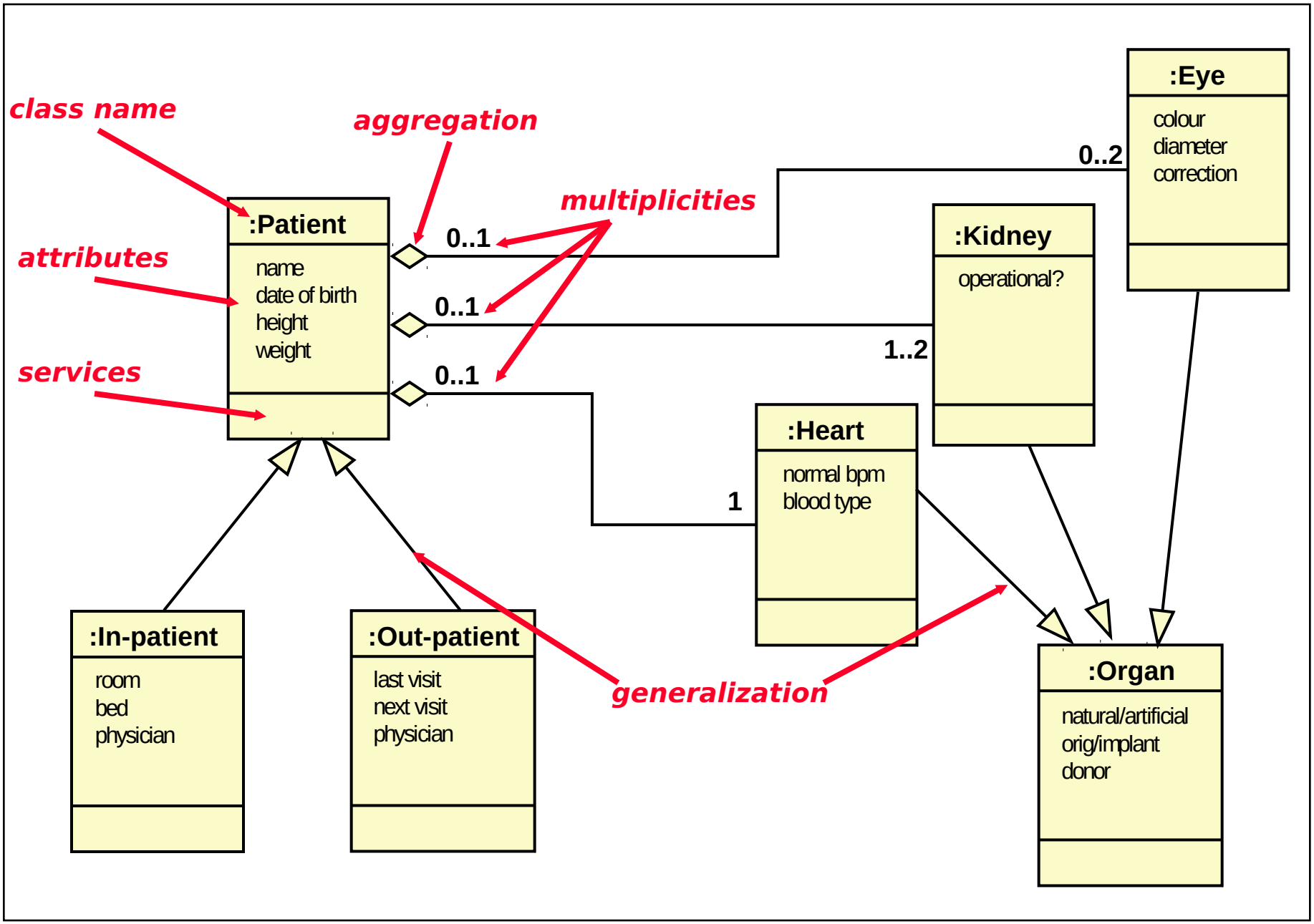
Source: *Adapted from Davis, 1990, p67-68*

Generalization (an abstraction hierarchy)

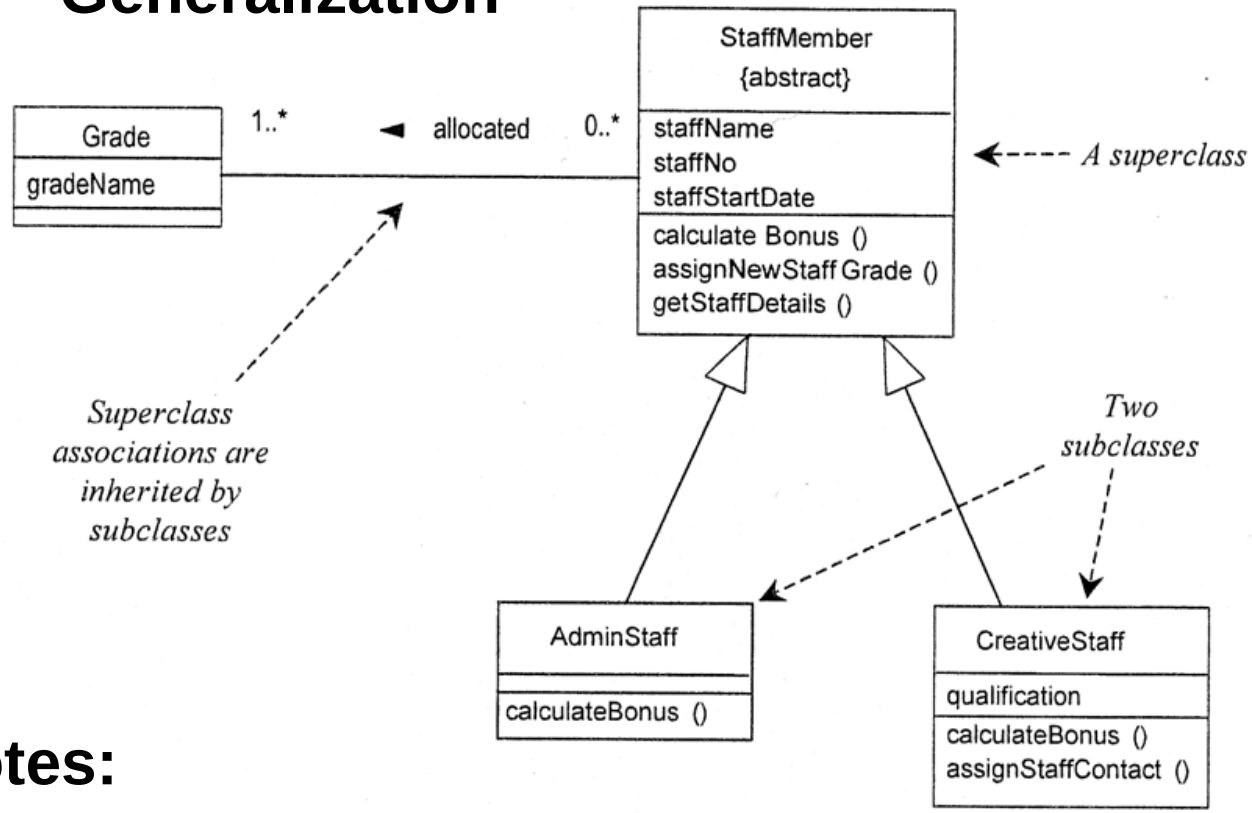


Aggregation (a partitioning hierarchy)




class name
aggregation
multiplicities
attributes
services
: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
generalization
:Organ
natural/artificial
orig/implant
donor

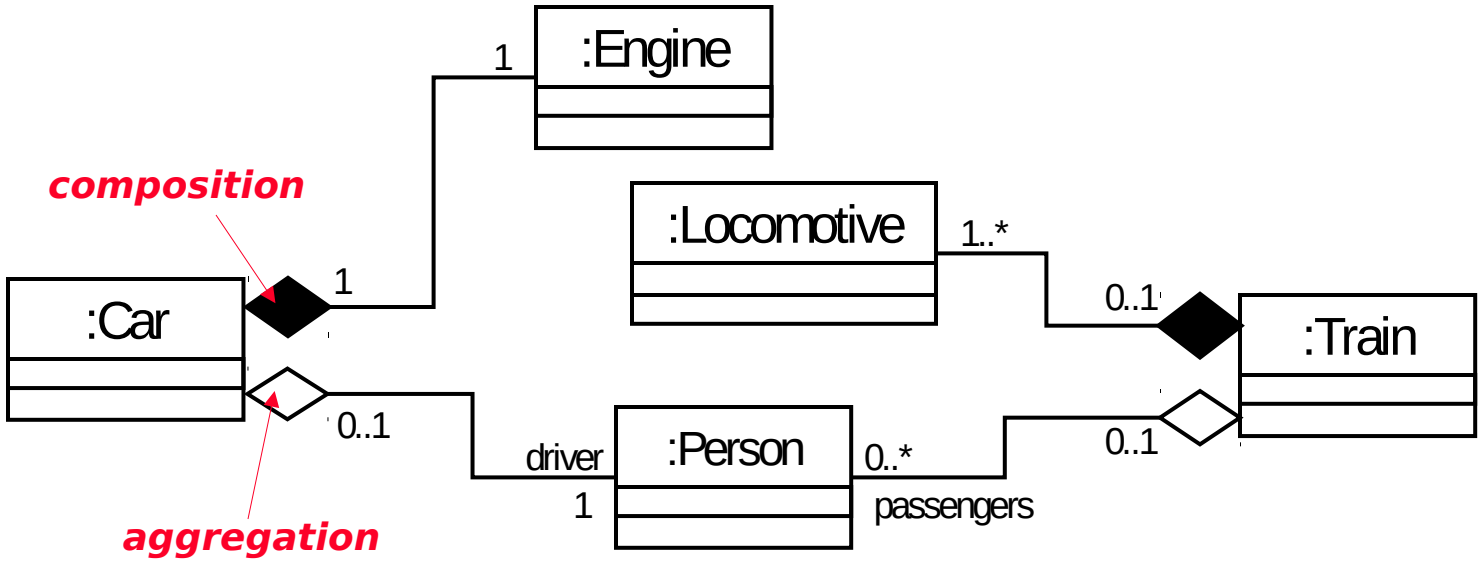
Generalization



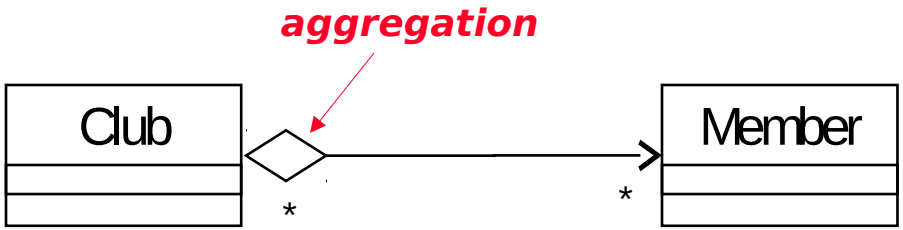
- **Notes:**
  - Subclasses inherit attributes, associations, & operations from the superclass
  - A subclass may override an inherited aspect
    - e.g. AdminStaff & CreativeStaff have different methods for calculating bonuses
  - Superclasses may be declared {abstract}, meaning they have no instances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

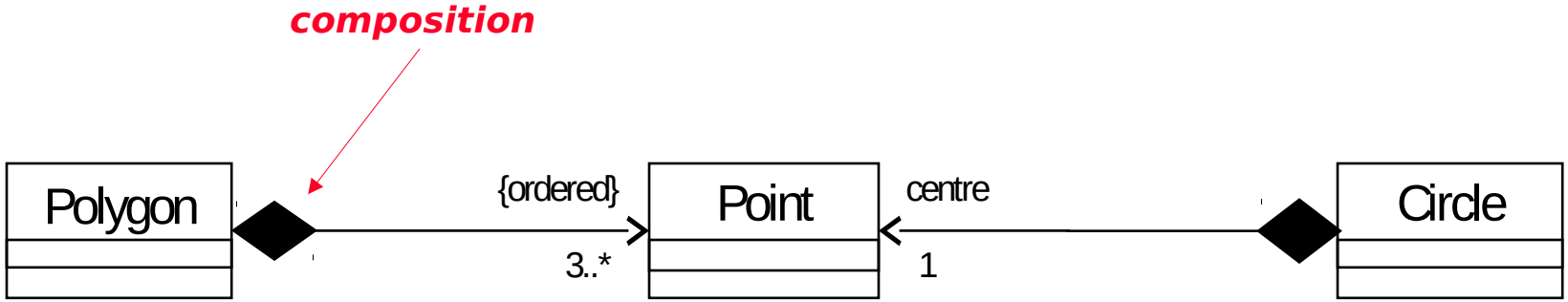
Aggregation and Composition

- **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



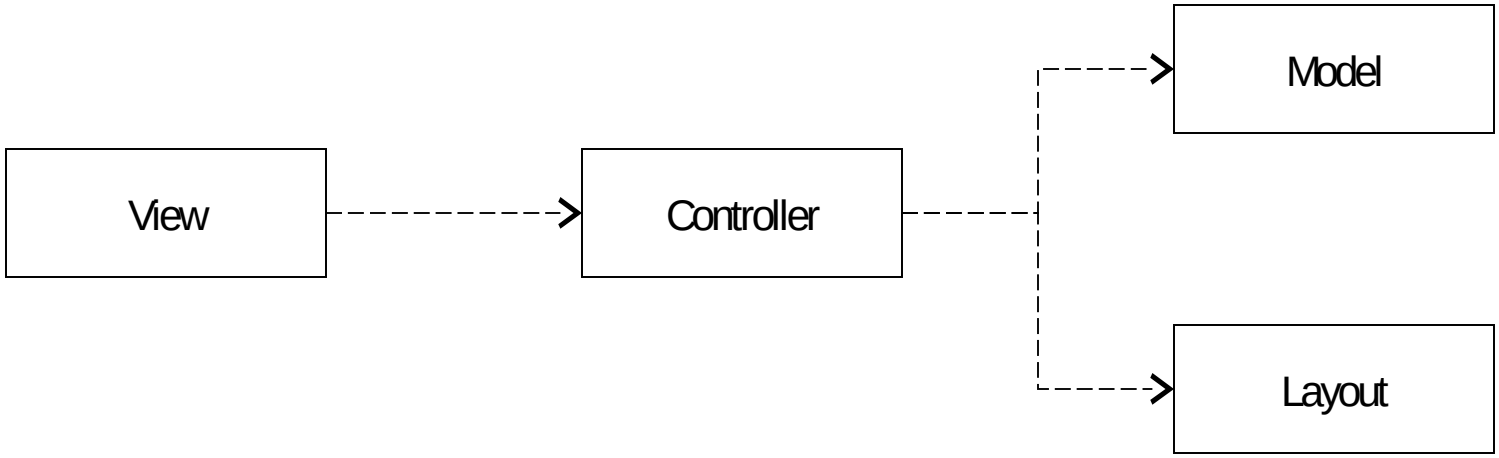
Aggregation / Composition





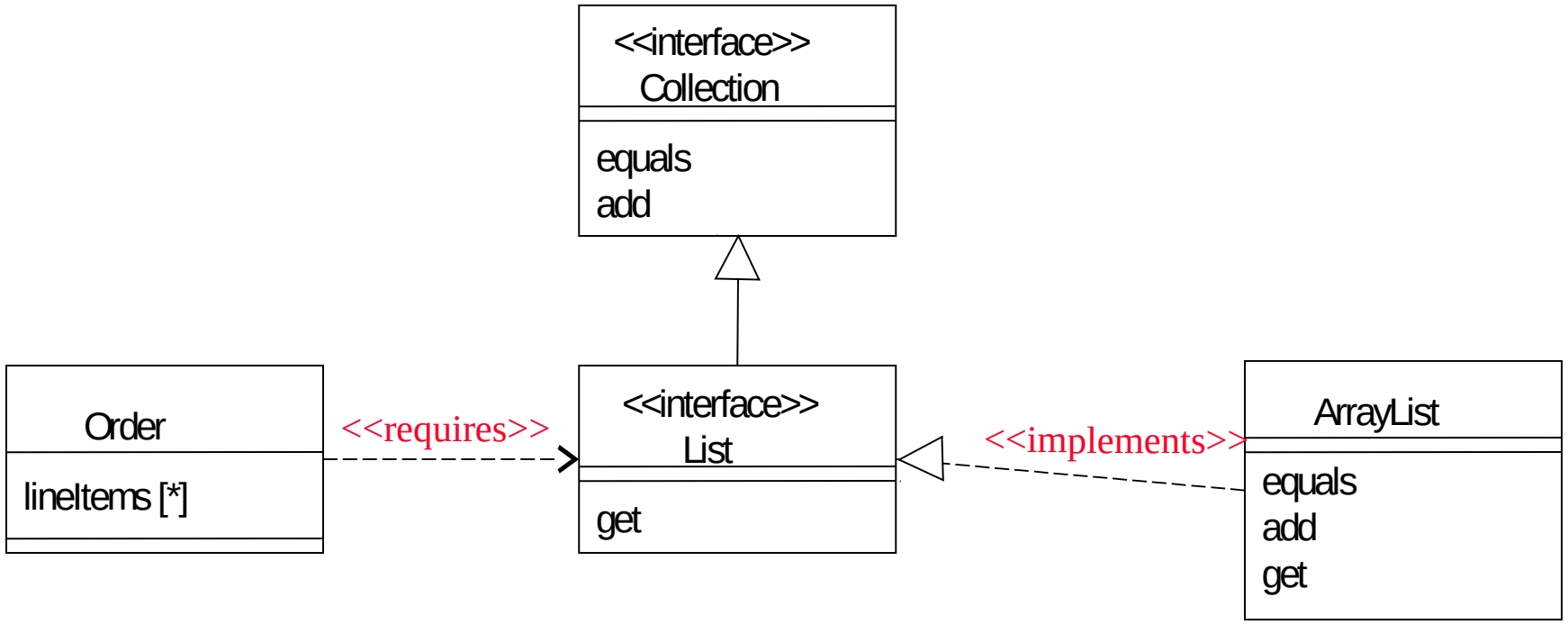
Note: No sharing - any instance of point can be part of a polygon or a circle, but not both

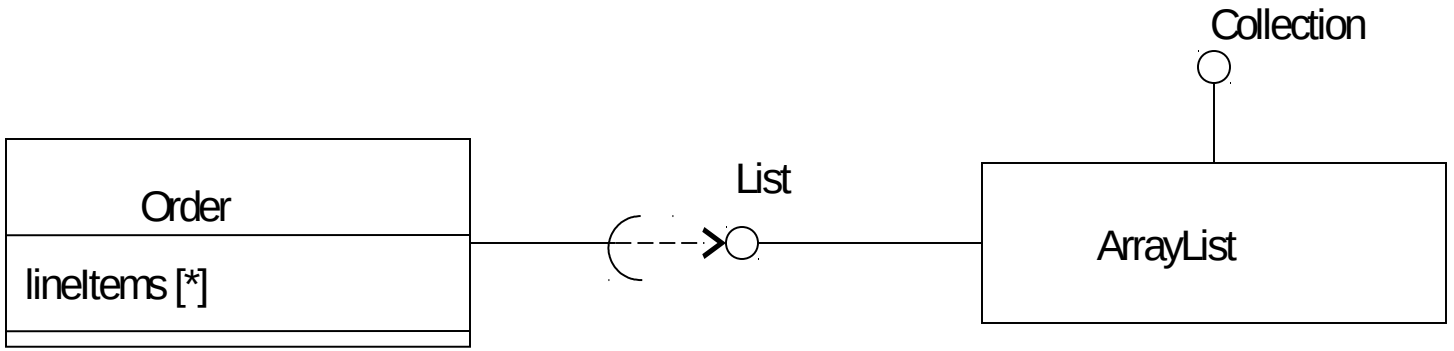
Dependencies



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

Interfaces

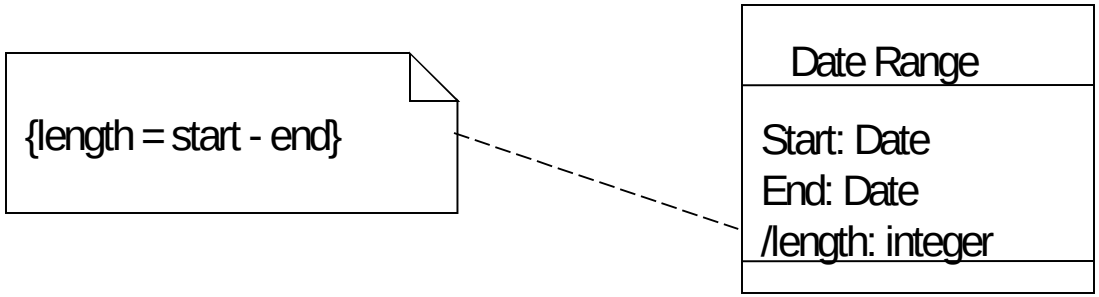

<<interface>>
Collection
equals
add
Order
lineItems [*]
<<requires>>
<<interface>>
List
get
<<implements>>
ArrayList
equals
add
get


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

Exercise

Recall our example from the lecture on CRC cards:

You need to develop a mobile application that helps school teachers and students on field trips. Students use this app to record their observations during a field trip. Teachers can preload the app with helpful templates suitable for a particular field trip. They can either choose a template from an existing bank of templates, or they can create their own. Each item in the template corresponds to a particular observation. The system also provides the students with an opportunity to edit their observations after the trip, prior to submission deadline. At the deadline (set by a teacher), the app sends the students' observations to the teacher for grading. In addition to assigning an overall grade, a teacher can leave a comment for a specic student's observation. Once graded, the work can be returned back to the student.

Create a UML model based on the CRC model.