Object Oriented Modelling

Department of Software Engineering, CIT,
Cantho University

Contents

- **Object Oriented Analysis**
  - Identifying Classes
  - Attributes and Operations
- **UML Class Diagrams**
  - Associations
  - Multiplicity
  - Aggregation
  - Composition
  - Generalization

Requirements & Domain Models

- Our analysis models should…
  - …represent people, physical things and concepts important to our understanding of what is going on in the application domain.
  - …show connections and interactions among these people, things and concepts.
  - …show the business situation in enough detail to evaluate possible designs.
  - …be organized to be useful later, during design and implementation of the software.
  - …allow us to check whether the functions we will include in the specification will satisfy the requirements.
  - …test our understanding of how the new system will interact with the world.

Object Oriented Analysis

- Background
  - Model the requirements in terms of objects and the services they provide.
  - Develop from object oriented design
    - Applied to modelling the application domain rather than the program.
- Motivation
  - OO is (claimed to be) more ‘natural’
    - As a system evolves, the functions it performs need to be changed more often than the objects on which they operate…
    - …a model based on objects (rather than functions) will be more stable over time…
    - …hence, object-oriented designs are more maintainable
  - OO emphasizes importance of well-defined interfaces between objects.

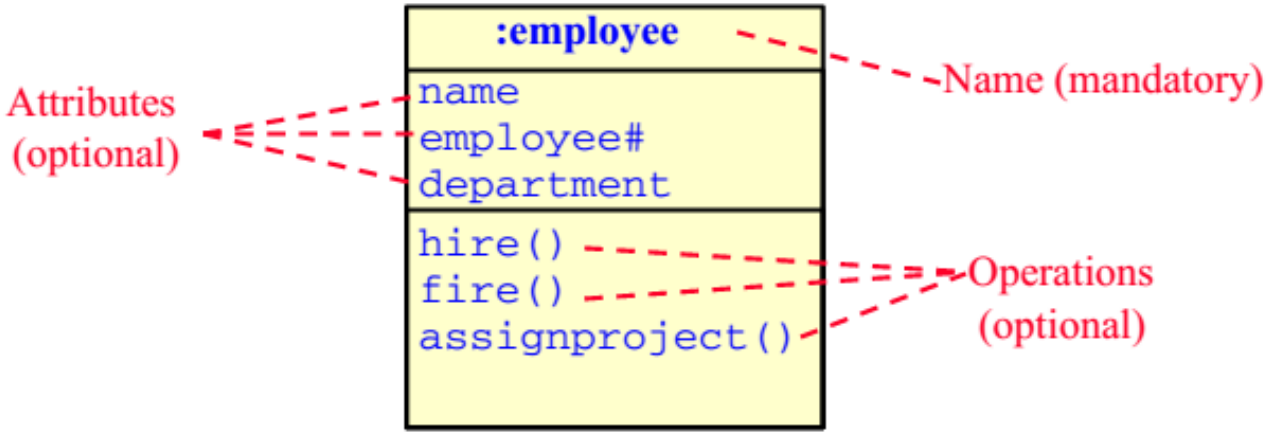
NOTE: OO applies to requirements engineering because it is a modeling tool. But we are modeling domain objects, not the design of the new system

Nearly anything can be an object

- External Entities
  - …that interact with the system being modeled
    - E.g. people, devices, other systems
- Things
  - …that are parts of the domain being modeled
    - E.g. reports, displays, signals, etc.
- Roles
  - played by people who interact with the system

What are classes?

- A class describes a group of objects with
  - similar **properties** (**attributes**),
  - common **behavior** (**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.



Finding Classes

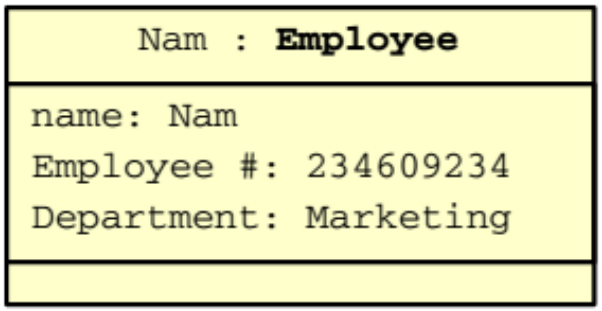
- Finding classes from source data:
  - Look for **nouns** and **noun phrases** in stakeholders' descriptions of the problem.
- Finding classes from other sources:
  - Reviewing background information;
  - Users and other stakeholders;
- It's better to include many candidate classes at first
  - You can always eliminate them later if they turn out not to be useful.

Selecting Classes

- Discard classes for concepts which:
  - Are beyond the scope of the analysis.
  - Duplicate other classes.
  - Are too vague or too specific.
    - e.g. have too many or too few instances

Objects vs. Classes

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



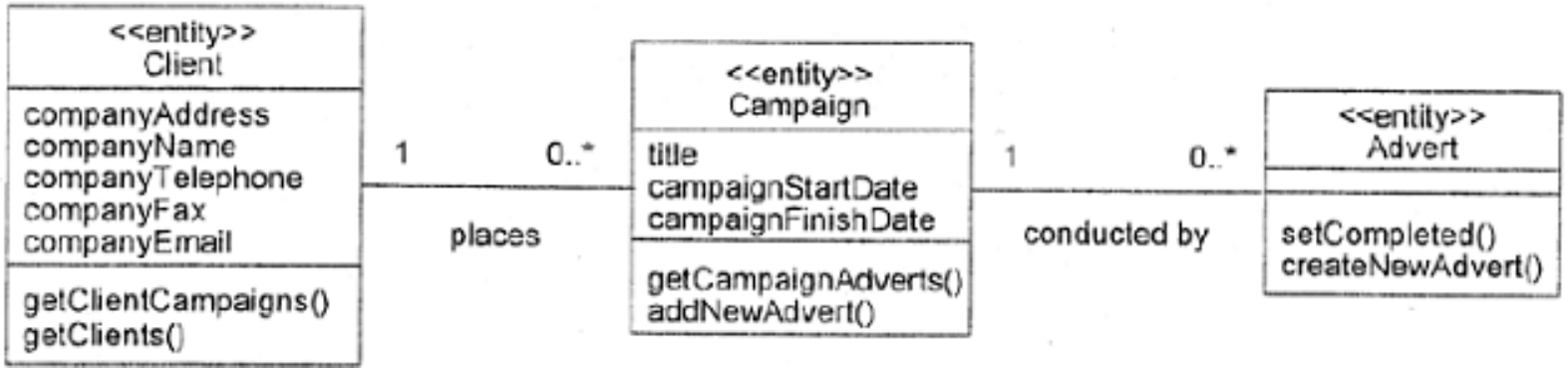
  - Two different objects may have identical attribute values (like two people with identical name and address)
- Objects have associations with other objects
  - E.g. Fred_Bloggs:employee is associated with the KillerApp:project object.
  - But we will capture these relationships at the class level.
  - **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??)

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**
  - **Note**: The last two are not useful during requirements analysis.

Associations (cont.)

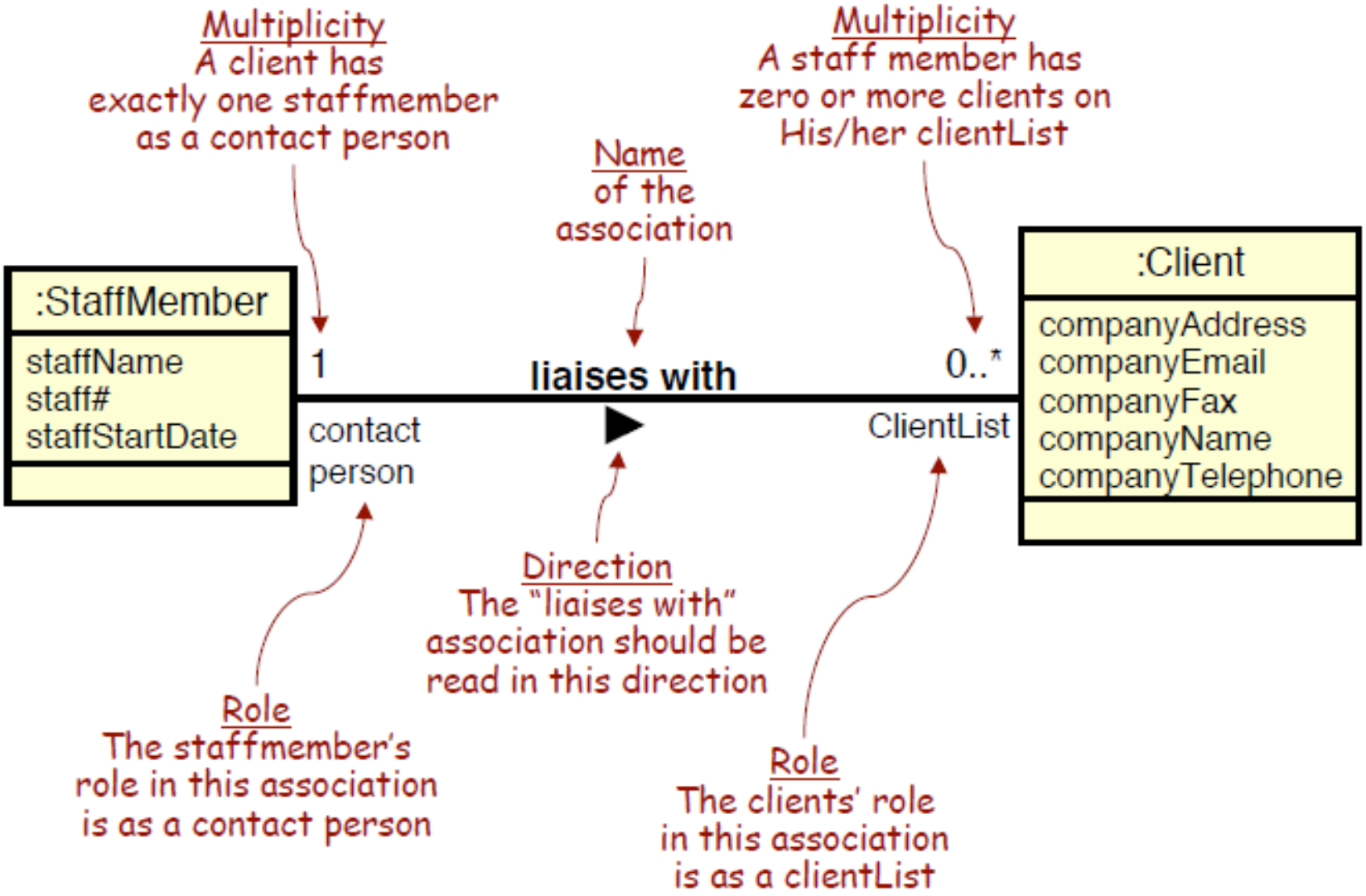
- Class diagrams show classes and their relationships



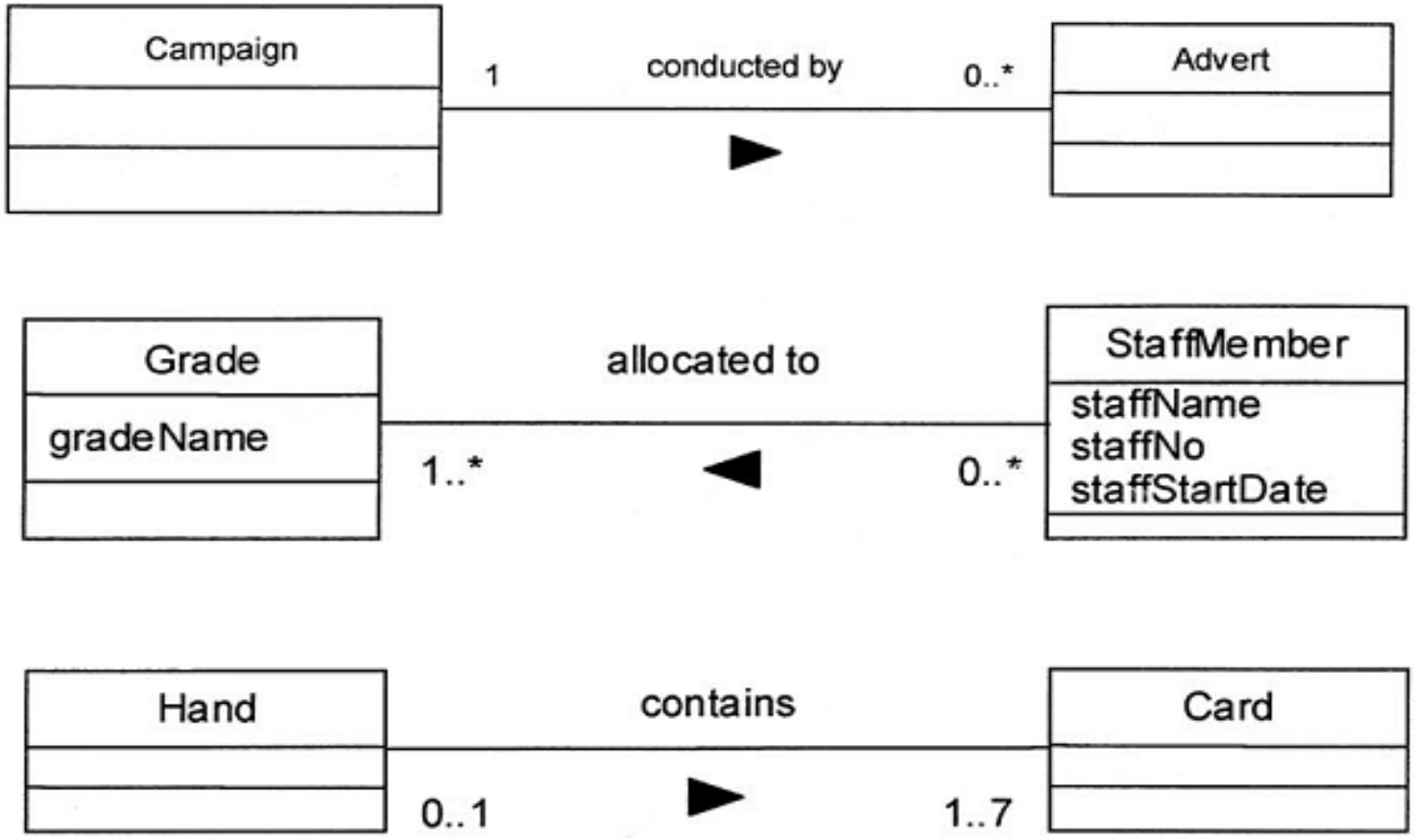
Association Multiplicity

- Ask questions about the associations:
  - Can a campaign exist without a member 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**)
- 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

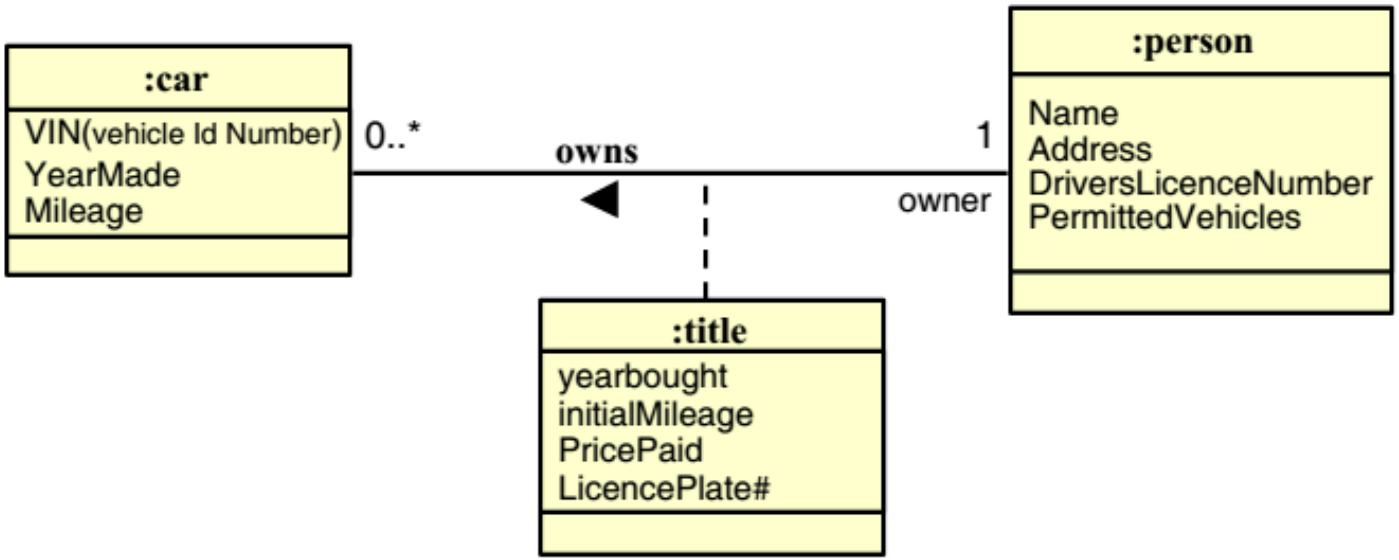


More Examples



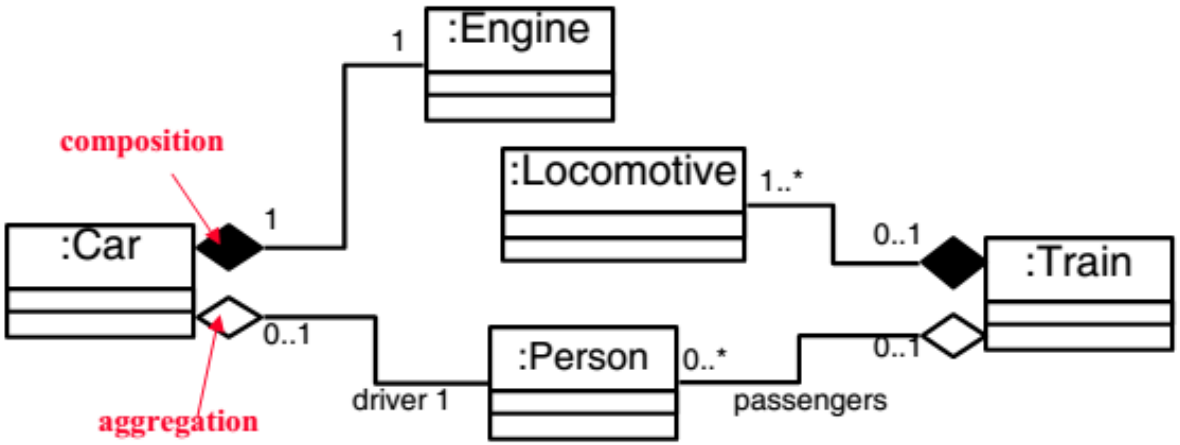
Association Classes

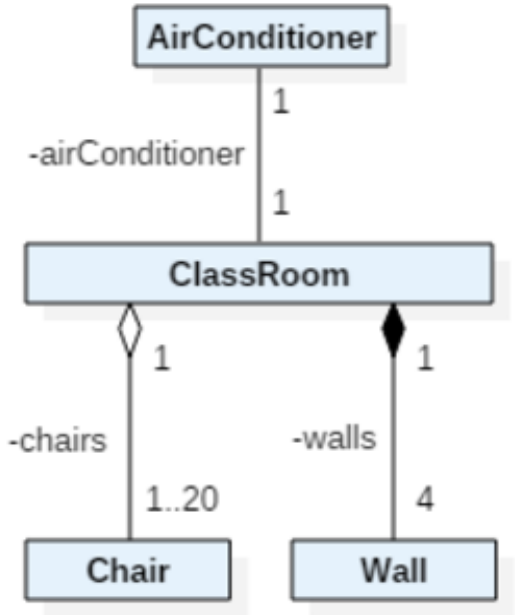
- Sometimes the association is itself a class
  - …because we need to maintain information about the association.
  - …and that information doesn't naturally live in the classes.
    - E.g. a "title" is an object that represents information about the relationship between an owner and her car



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.

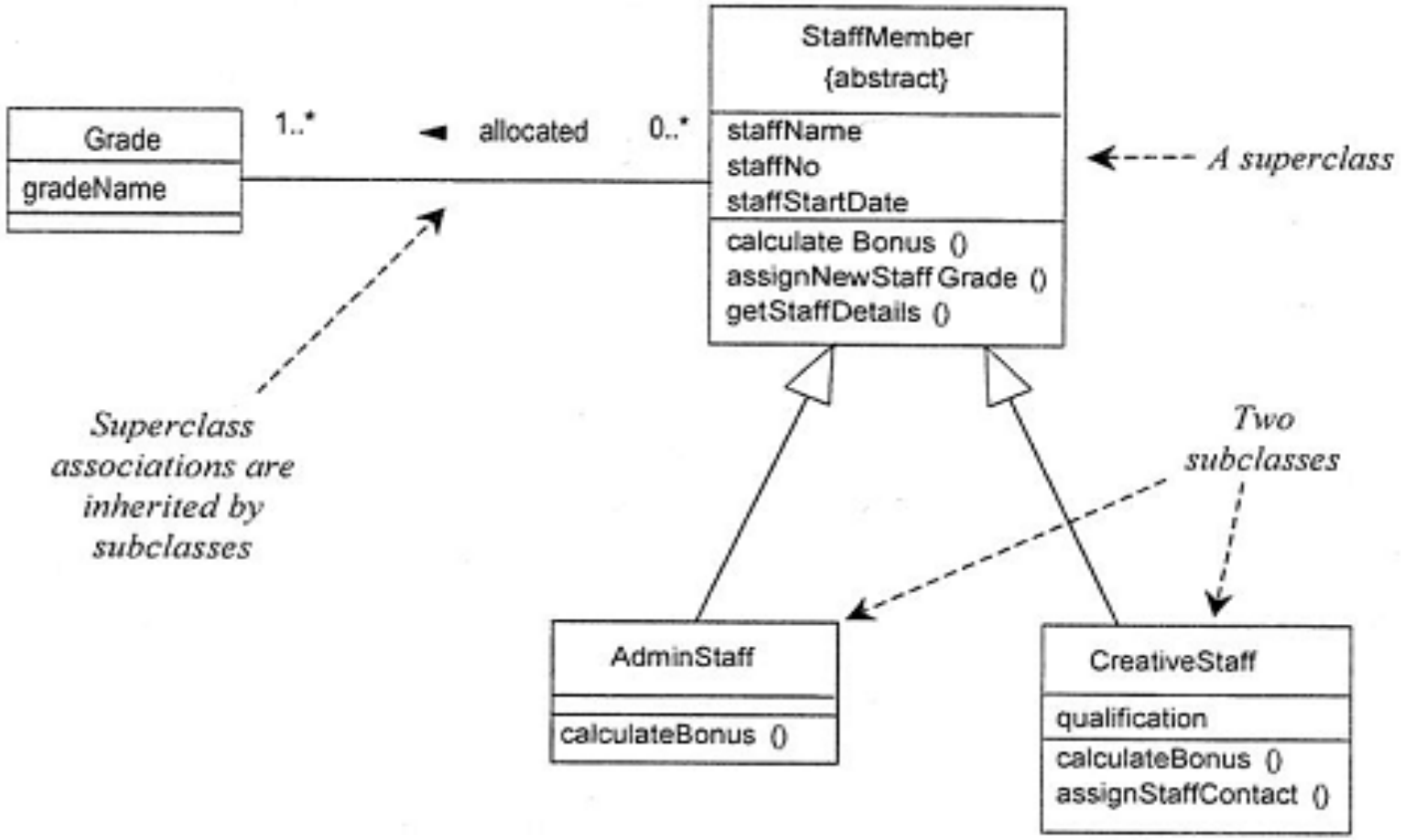




Generalization

- 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**

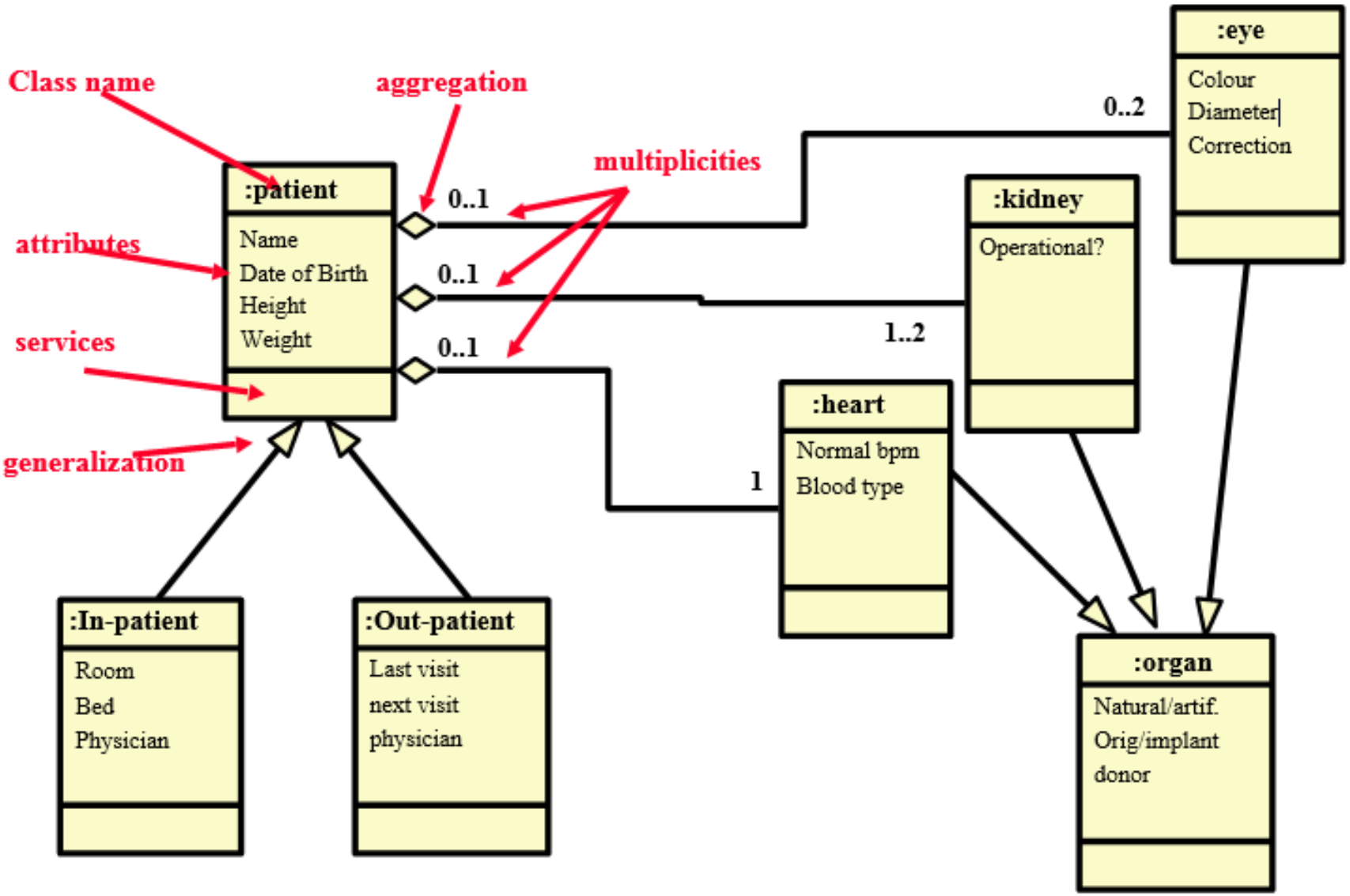
Generalization (cont.)



Class Diagrams

- A collection of class diagrams represent the whole system.
- Class diagrams show different objects that are going to be used in systems and their relationships.
- Important elements:
  - **Class**
  - **Relationships (such as Inheritance, Composition, Aggregation, Association, Dependency)**
  - OCL (Object Constraint Language: Invariant, Precondition, Postcondition)
  - Interfaces

Class Diagrams



Main references

- Prof Steve Easterbrook, lecture notes, University of Toronto, Canada.
- https://github.com/imalitavakoli/learn-uml2

Q&A