Object Oriented

UML Class Diagram

Remember from the midterm . . .

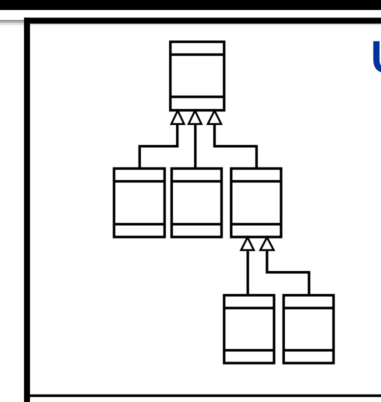
7 - Design a software system, in terms of interfaces, which contain headers of public methods, for the following problem statement: (20)

- Customers order products from an online store; their orders will be processes by the closest store to their address, and their bills will be issued. After the payment is done, items will be shipped to the customer address.

the Unified Modelling Language (UML)

- Standardized general-purpose modeling language
  - Used to specify, visualize, construct, and document the design of an object-oriented system under development
  - Offers a way to visualize various elements of a system such as activities, actors, business processes, database schemas, logical components, programming language statements, and reusable software components.
  - Combines techniques from data modeling(entity relationship diagrams), business modeling (work flows), object modeling, and component modeling
- Booch, Rumbaugh & Jacobson are principal authors
  - Still evolving (currently version 2.3)
  - Attempt to standardize the proliferation of OO variants
- Is purely a notation
  - No modelling method associated with it!
  - Was intended as a design notation
  - Can be used anywhere in the software development cycle
- 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, …

Modeling Notations

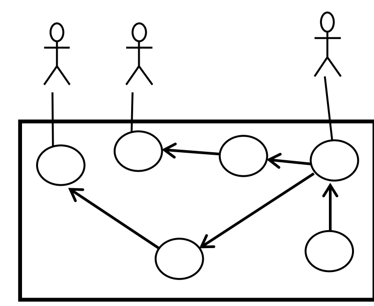


UML Class Diagrams

information structure

relationships between data items

modular structure for the system

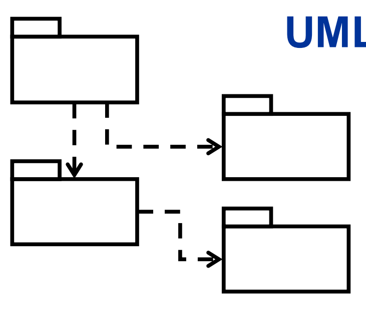


Use Cases

user's view

Lists functions

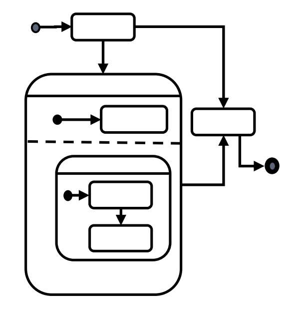
visual overview of the main requirements



UML Package Diagrams

Overall architecture

Dependencies between components

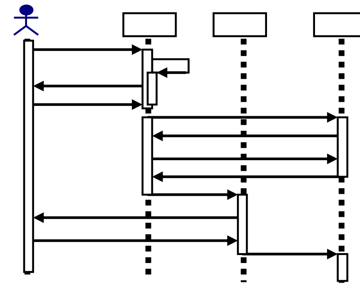


(UML) Statecharts

responses to events

dynamic behavior

event ordering, reachability, deadlock, etc

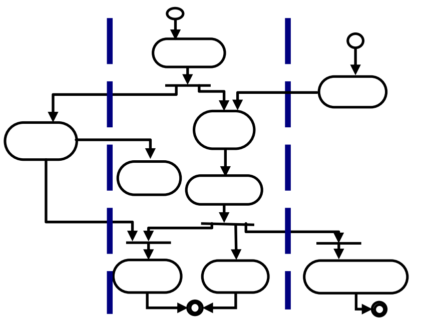


UML Sequence Diagrams

individual scenario

interactions between users and system

Sequence of messages



Activity diagrams

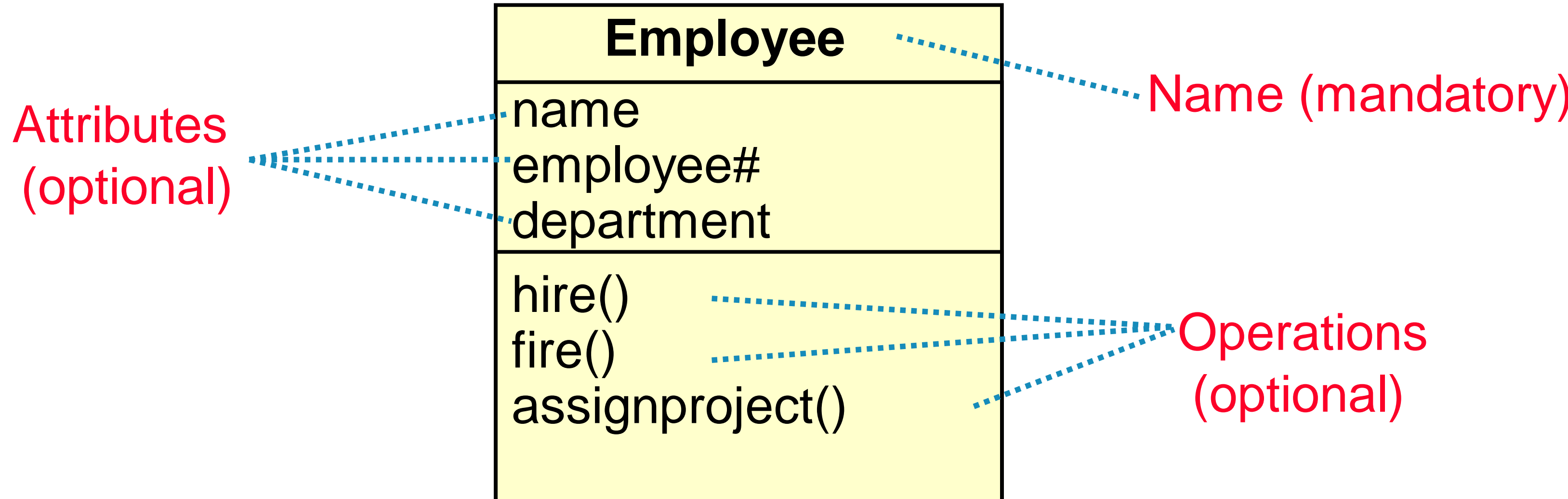
business processes;

concurrency and synchronization;

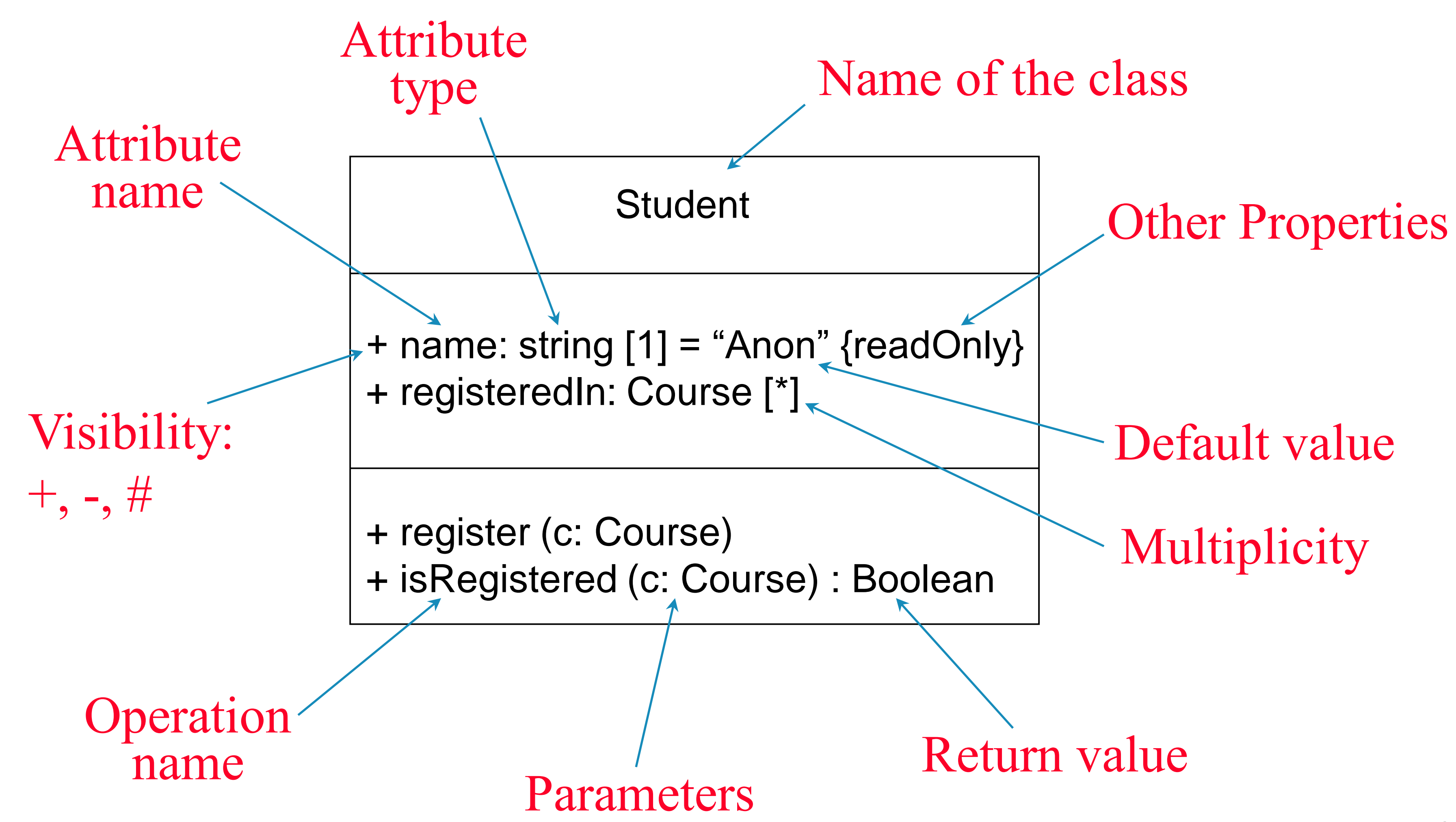
dependencies between tasks;

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

  - **The relation between an Object and its Class is called "*Instantiation*"**
  - **Two different objects may have identical attribute values (like two people with identical name and address)**
  - **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??)**

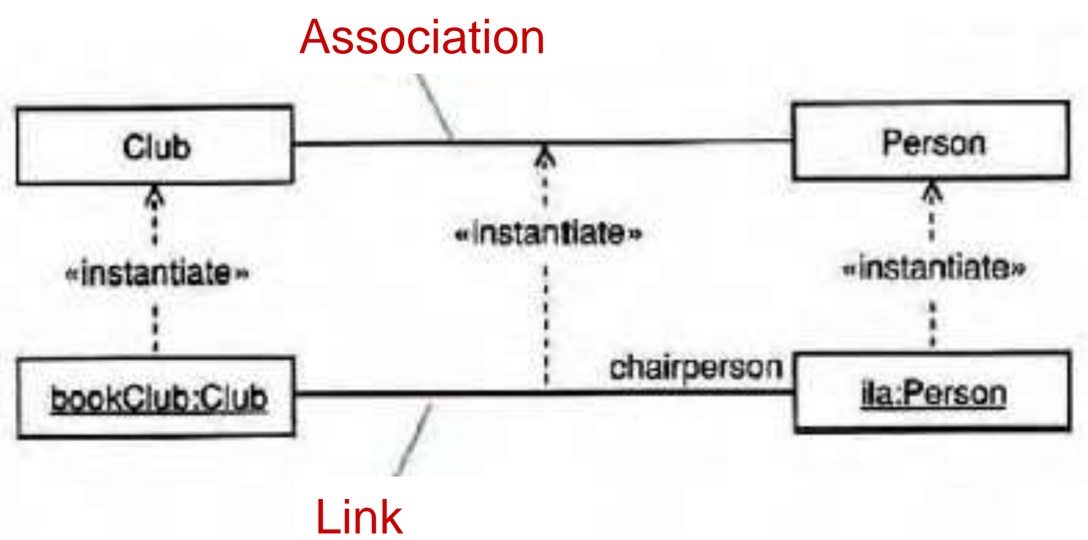
Relationships

- Objects do not exist in isolation from one another
  - A relationship represents a connection among things.
  - E.g. Fred_Bloggs:employee is associated with the KillerApp:project object
  - But we will capture these relationships at the class level (why?)

- Class diagrams show classes and their relationships
  - In UML, there are different types of relationships:
    - Association
    - Aggregation and Composition
    - Generalization
    - Dependency
    - Realization

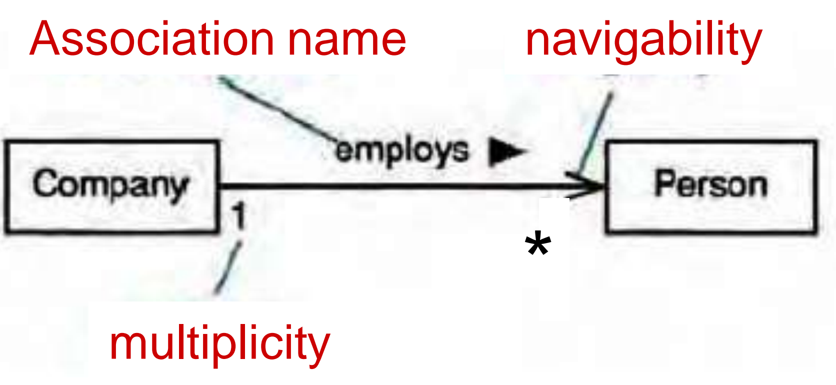
Association

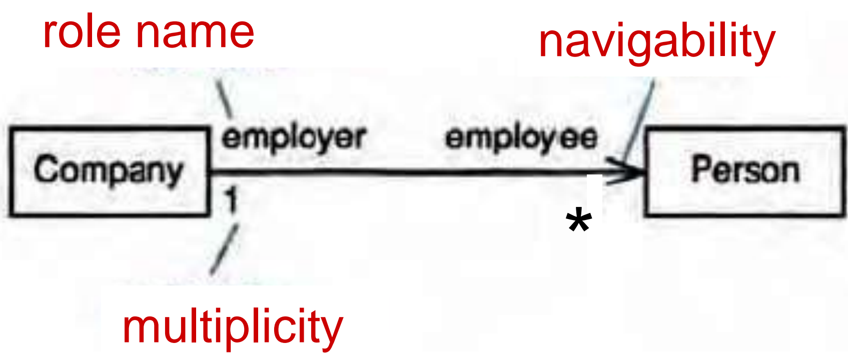
- Associations are semantic connections between classes.
  - If there is a link between two objects, there must be an association between the classes of those objects.
  - Links are instances of associations just as objects are instances of classes.



Association

- Associations may optionally have the following:
  - Association name
    - may be prefixed or postfixed with a small black arrowhead to indicate the direction in which the name should be read;
    - should be a verb or verb phrase;
  - Role names
    - on one or both association ends;
    - should be a noun or noun phrase describing the semantics of the role;
  - Multiplicity
    - The number of objects that can participate in an instantiated relation
  - Navigability





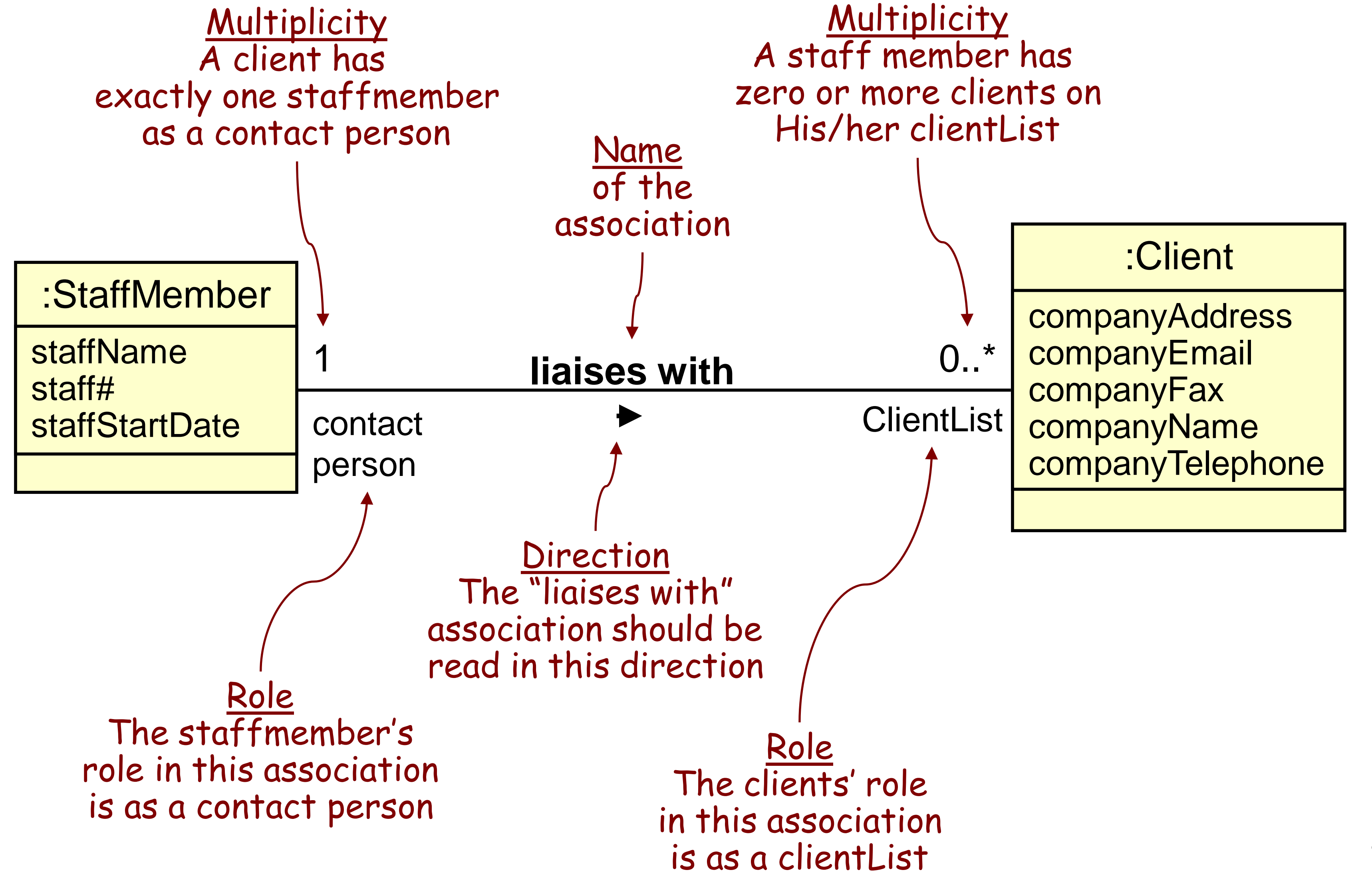
Association Multiplicity

- **Ask questions about the associations:**
  - **Can a company exist without any employee?**
    - If yes, then the association is optional at the Employee end - zero or more (0..*)
    - If no, then it is not optional - one or more (1..*)
    - If it must have only one employee - exactly one (1)
  - **What about the other end of the association?**
    - Can an employee work for more than one company?
    - No. So the correct multiplicity is one.

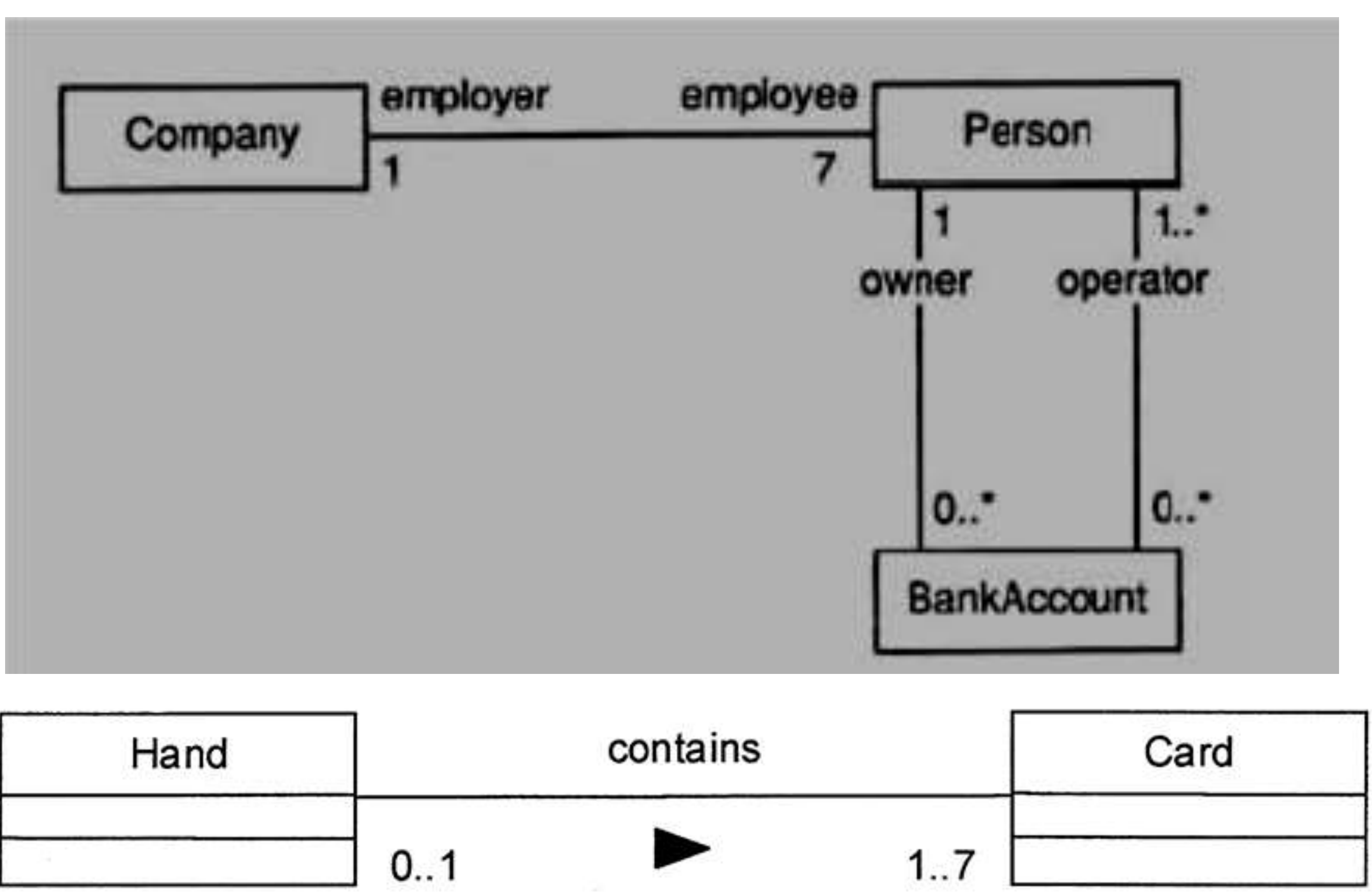
- **Some examples of specifying multiplicity:**

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

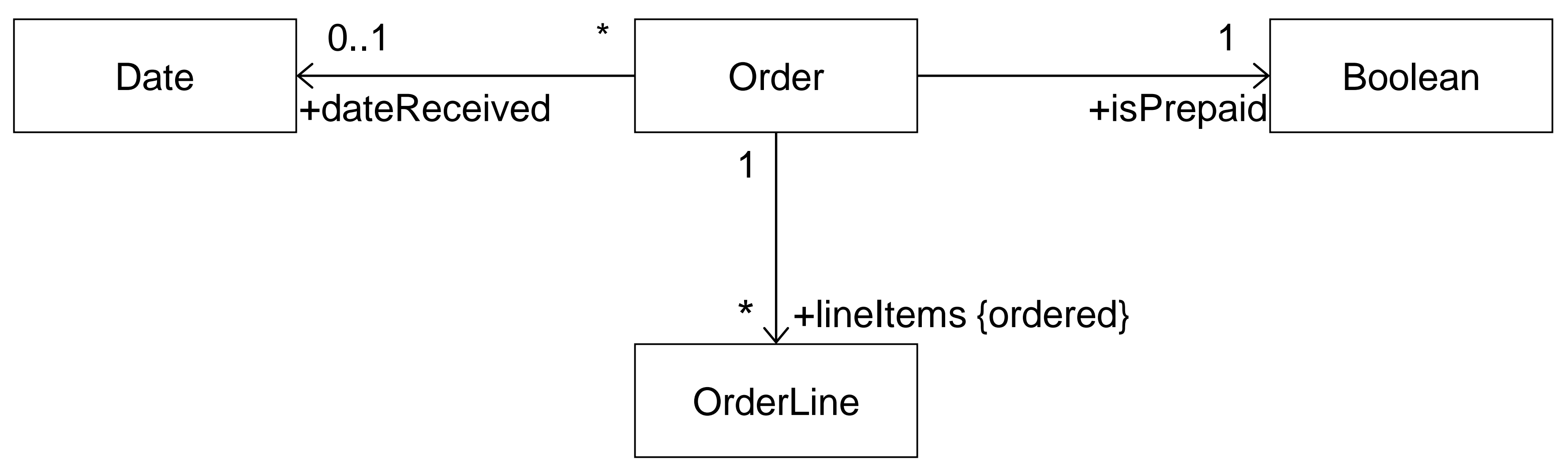


More Examples

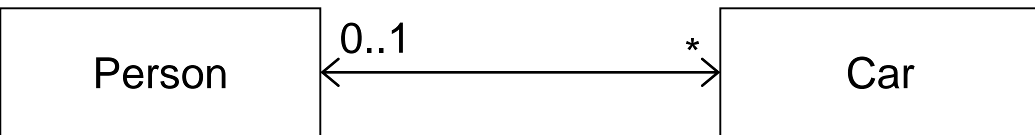


Navigability / Visibility

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



Bidirectional Associations



How implement it?

Person

+ carsOwned: Car [*]

Car

+ Owner: Person [0..1]

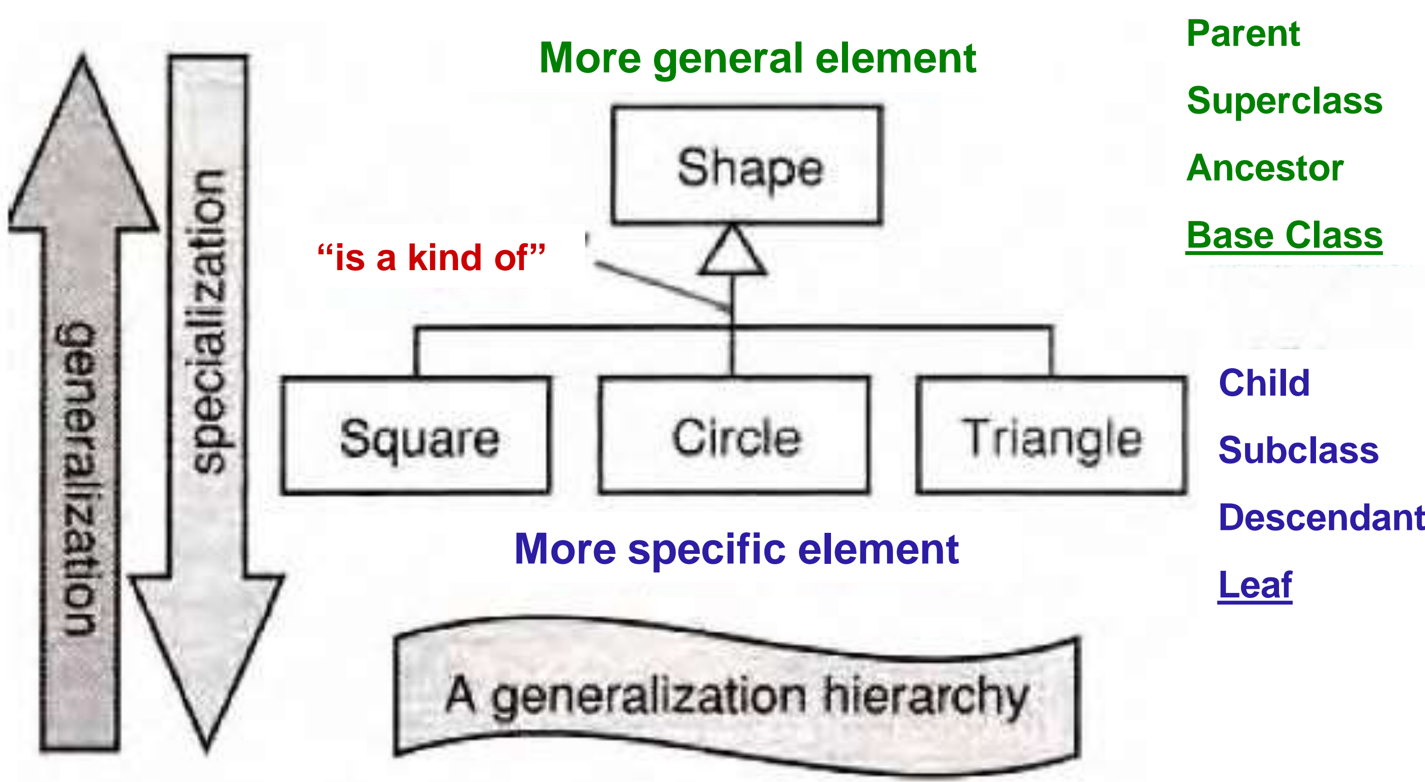
Implementation Complexities !

Generalization

- Generalization is a relationship between a more general thing and a more specific thing:
  - the more specific thing is consistent in every way with the more general thing.
  - the substitutability principle states that you can substitute the more specific thing anywhere the more general thing is expected.

Generalization/Specialization

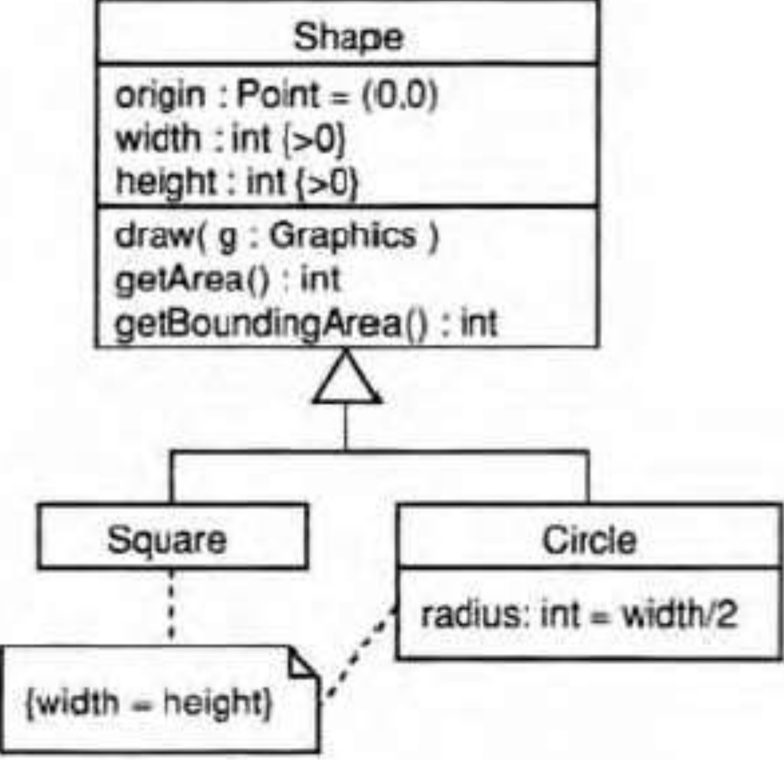
- Generalization hierarchies may be created by generalizing from specific things or by specializing from general things.



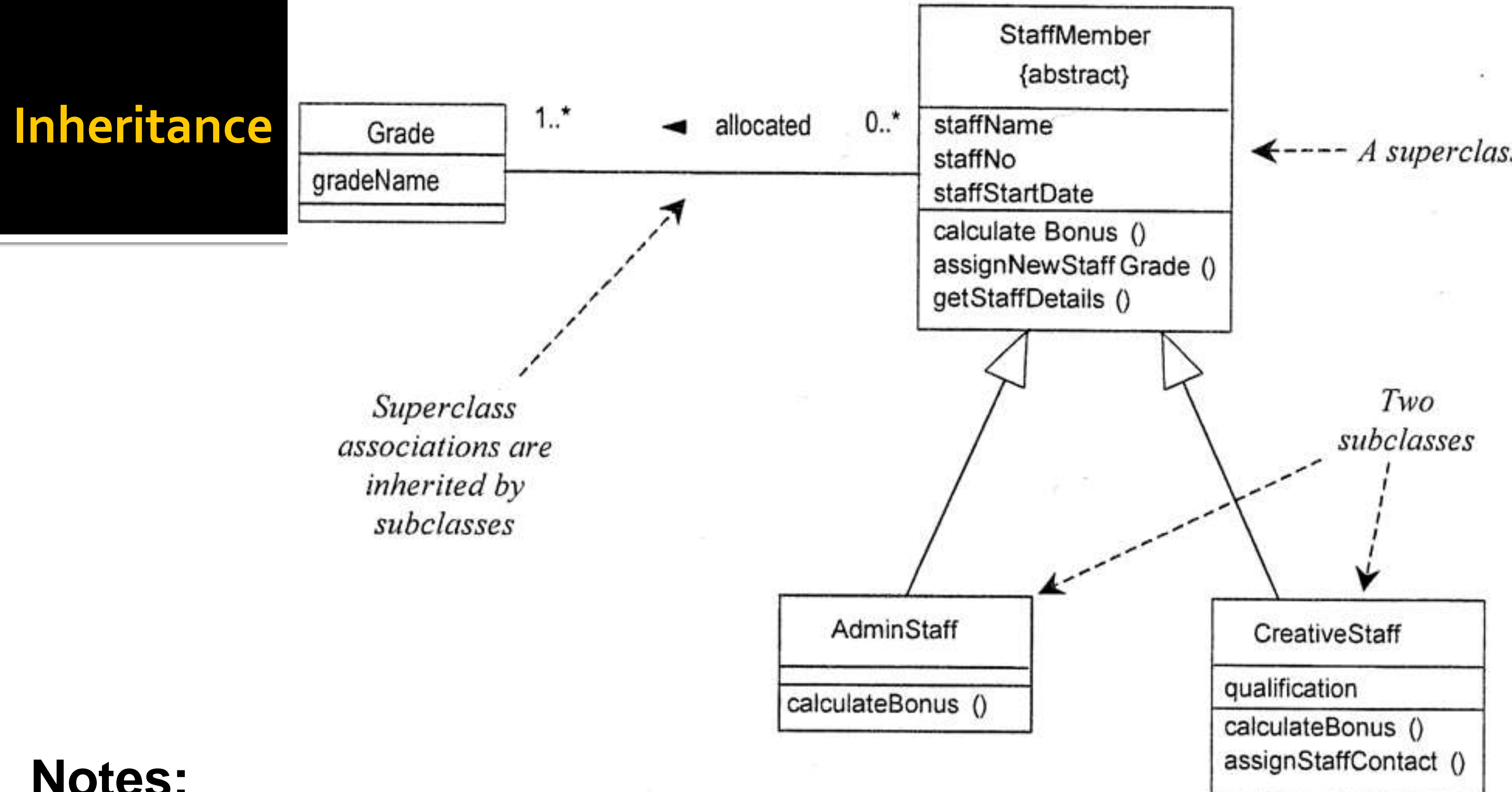
Inheritance

- **Class inheritance is implicit in a generalization relationship between classes.**
- **Subclasses inherit attributes, associations, & operations from the superclass**

What is the inheritance mechanism in Java?



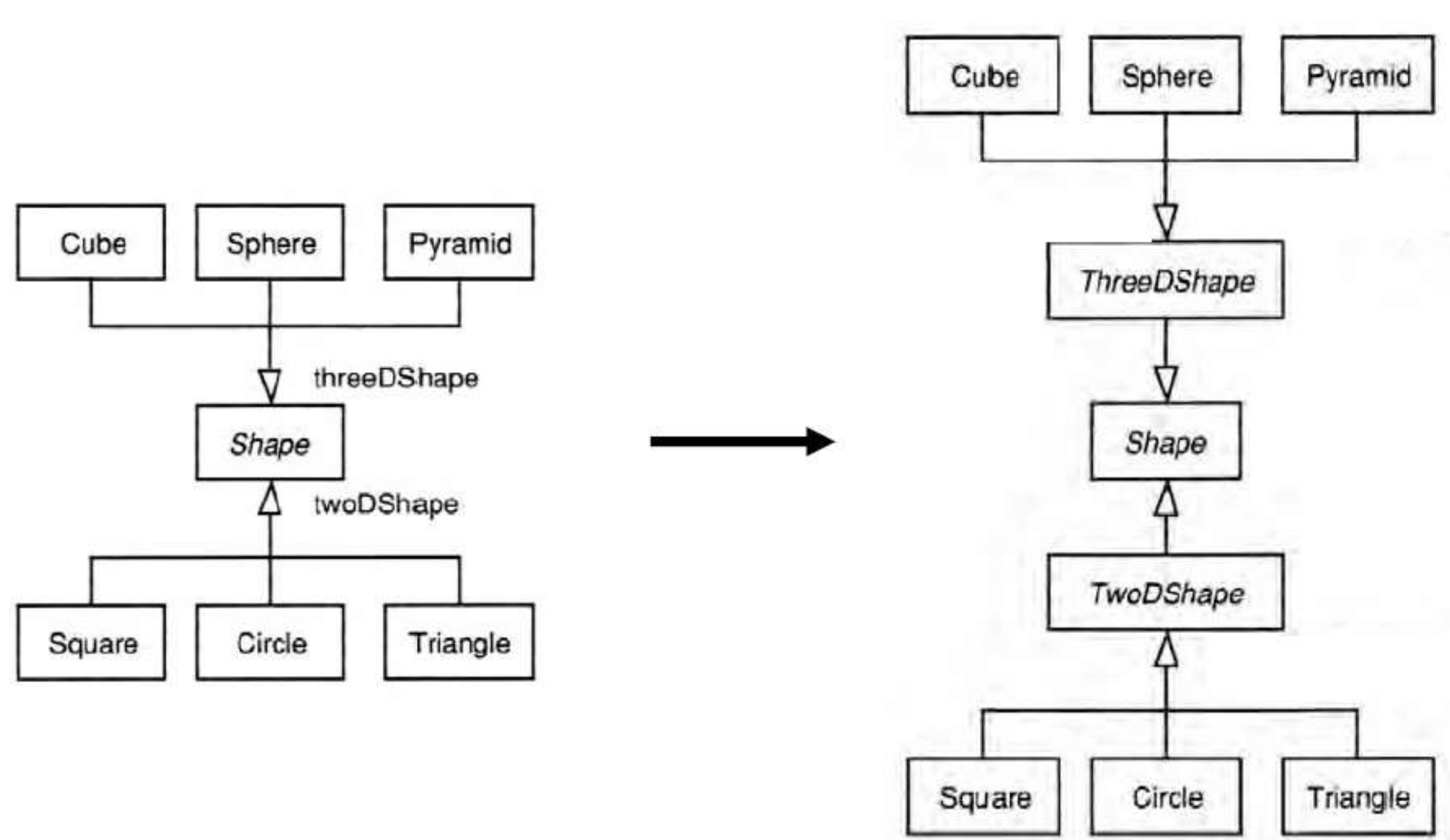
Inheritance



Notes:

- A subclass may override an inherited aspect
  - e.g. AdminStaff & CreativeStaff have different methods for calculating bonuses
- A Subclass may add new features
  - qualification is a new attribute in CreativeStaff
- 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

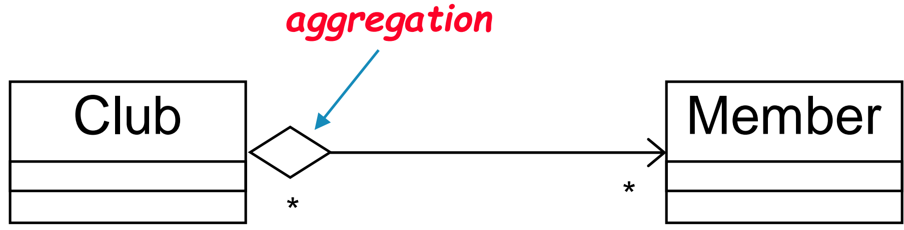
Generalization Sets: Implementation

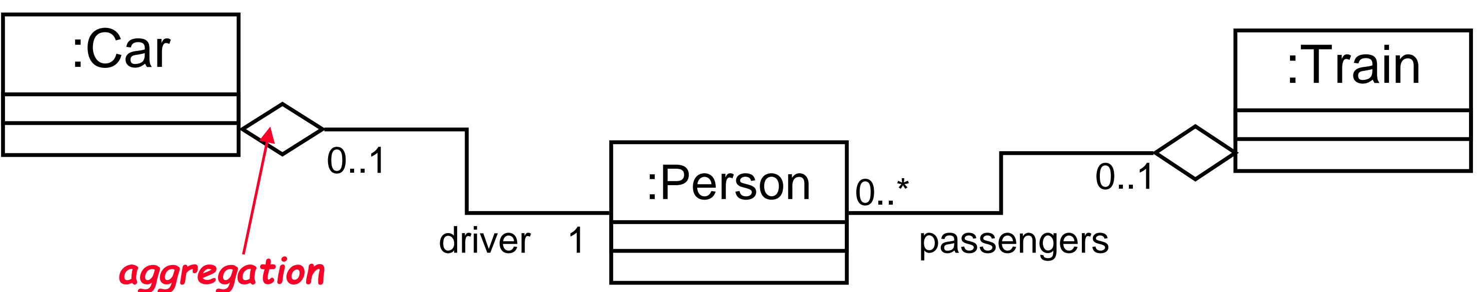


Aggregation and Composition

Aggregation

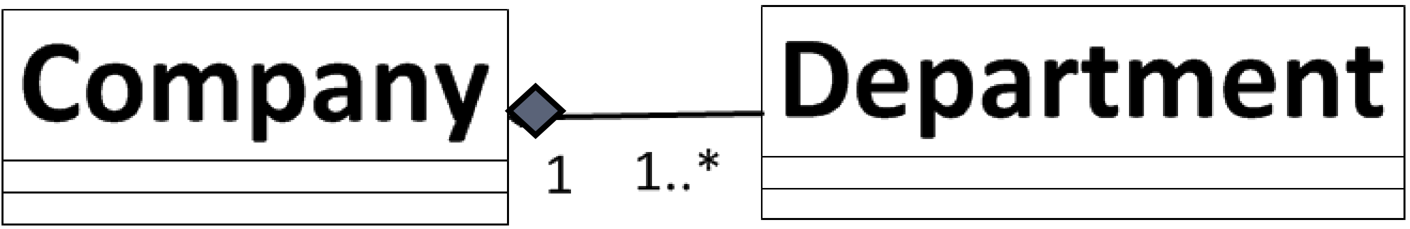
- This is the "**Has-a**" or "**Whole/part**" relationship

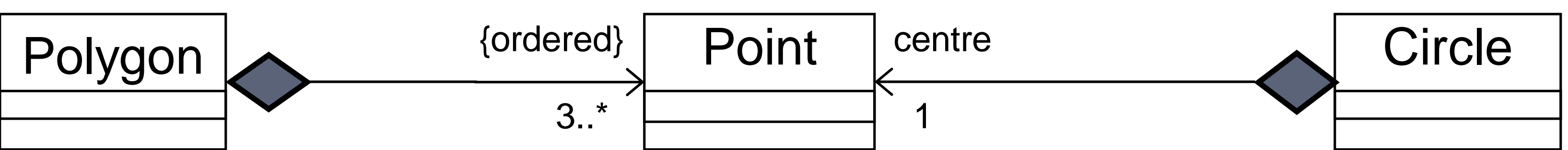




Aggregation and Composition

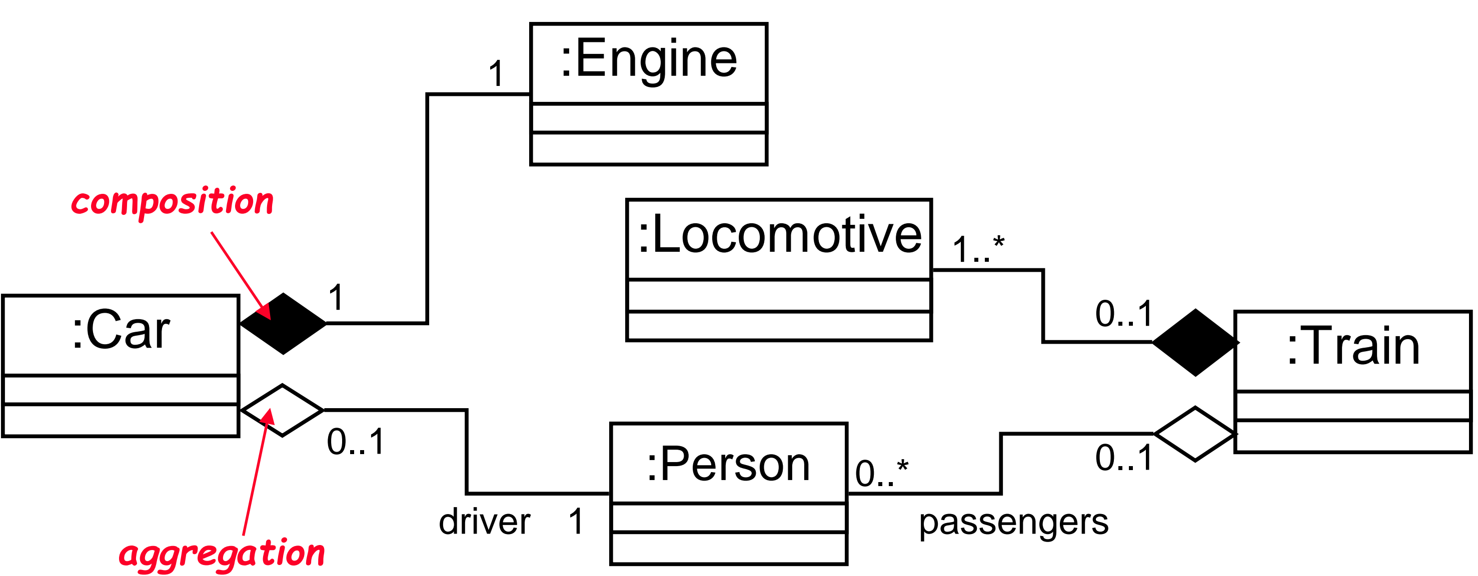
- **Aggregation**
  - This is the "Has-a" or "Whole/part" relationship
- **Composition**
  - Strong form of aggregation that implies ownership:
    - if the whole is removed from the model, so is the part.
    - the whole is responsible for the disposition of its parts
    - Note: Parts can be removed from the composite (where allowed) before the composite is deleted





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

Aggregation and Composition



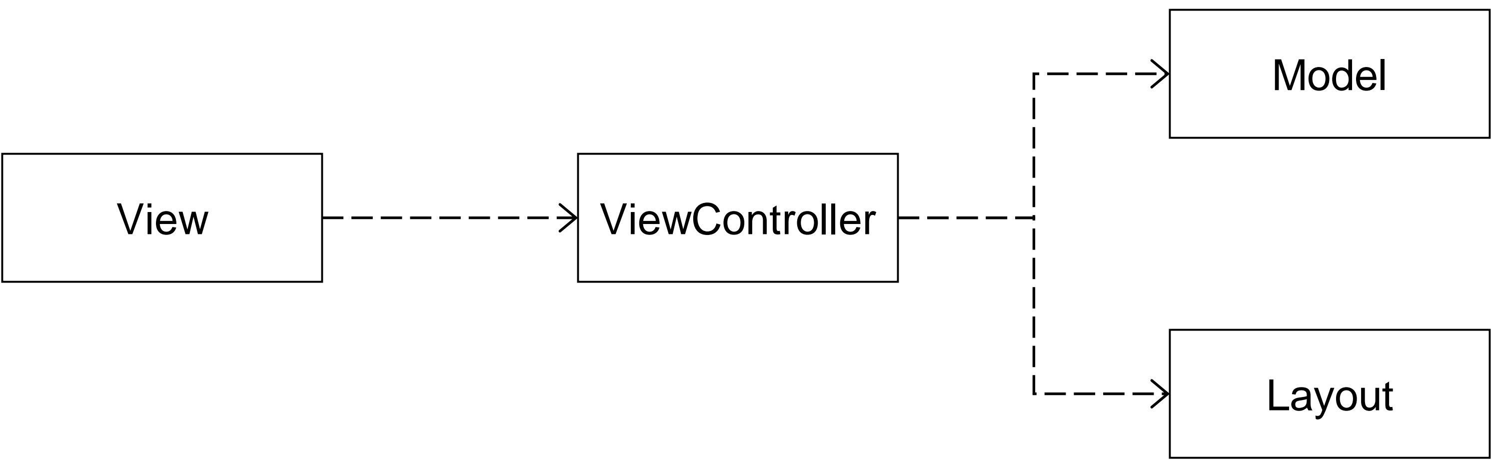
Class Activity

- Draw the UML class diagram which represents a file system – containing files and directories

Dependency

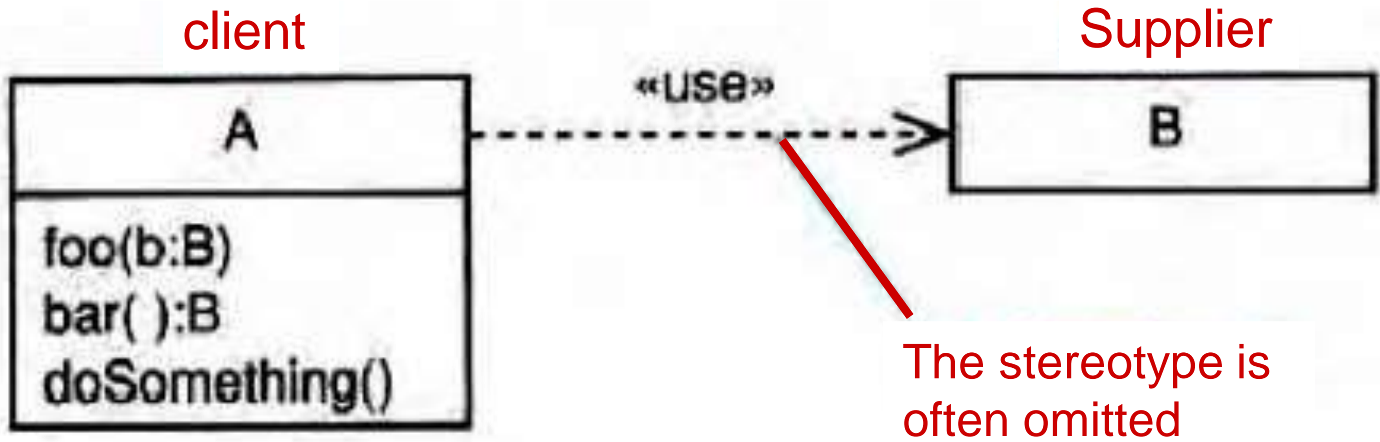
Dependencies are relationships in which a change to the supplier affects, or supplies information to, the client.

- The client depends on the supplier in some way.
- Dependencies are drawn as a dashed arrow from client to supplier.

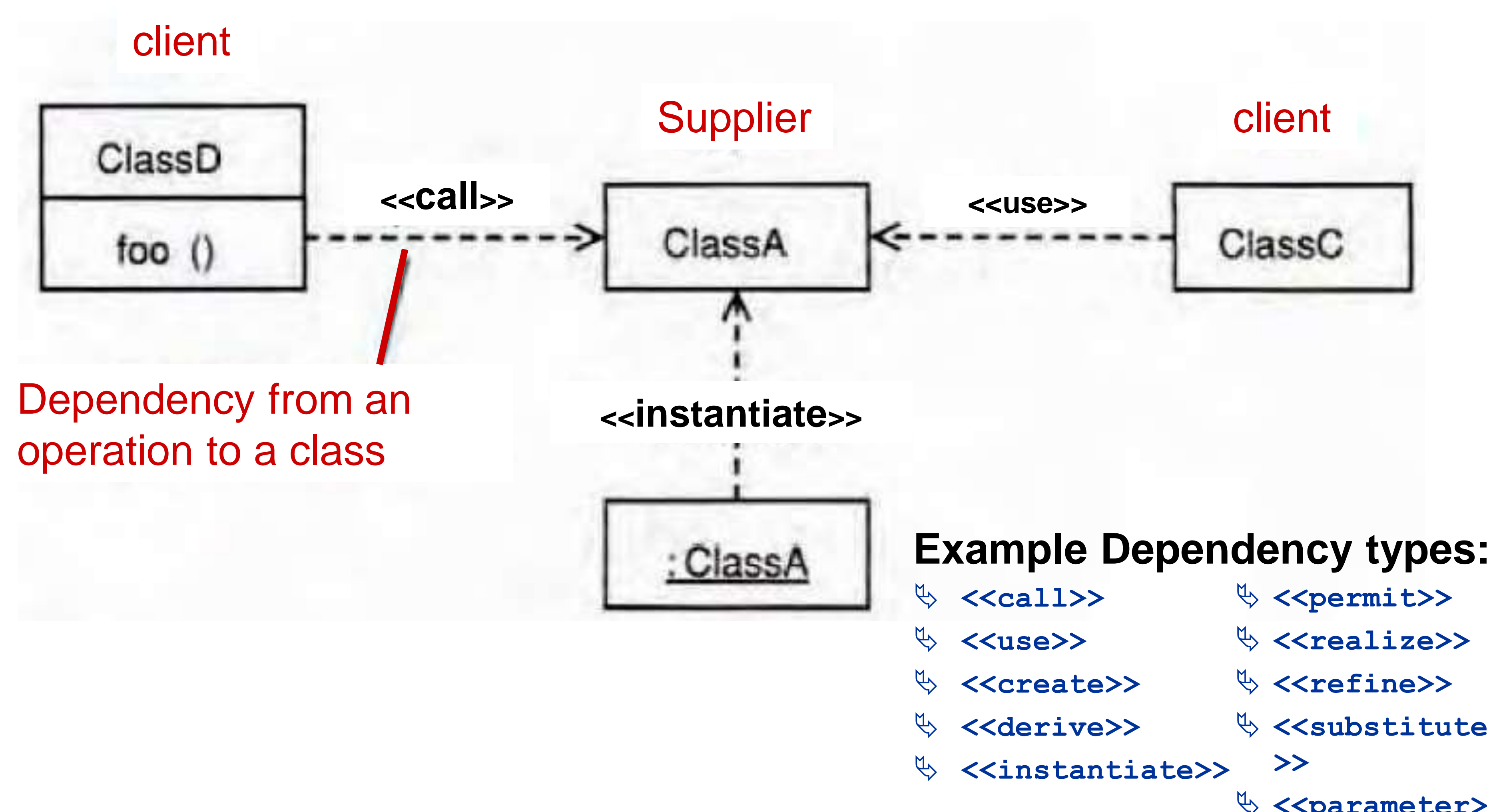


Usage Dependencies

- «use»-the client makes use of the supplier in some way -this is the catch-all.
- «call»-the client operation invokes the supplier operation.
- «parameter»-the supplier is a parameter or return value from one of the client's operations.
- «instantiate»-the client is an instance of the supplier.



Dependencies: Example



Interfaces

<<interface>>
Collection

equals
add

Order

LineItems [*]

<<requires>>

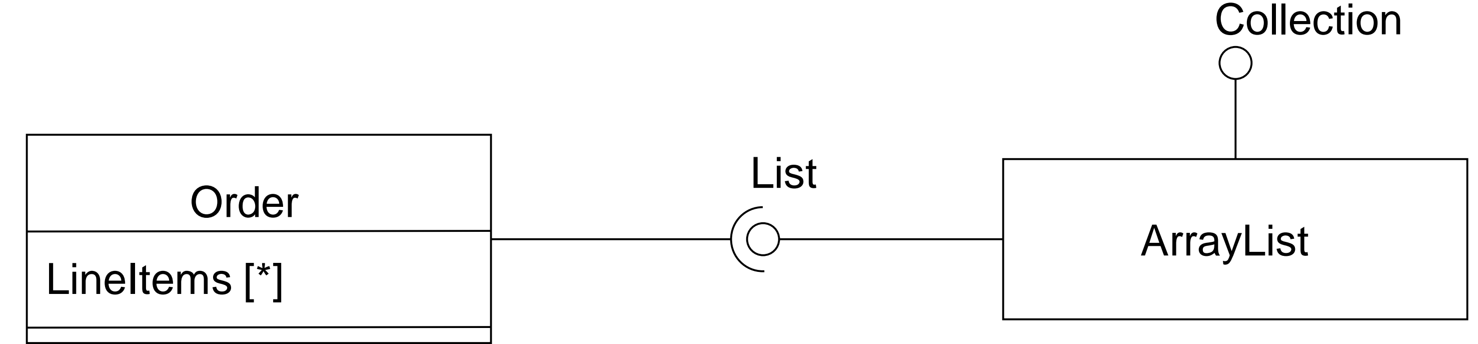
<<interface>>
List

get

<<realizes>>

ArrayList

get
add

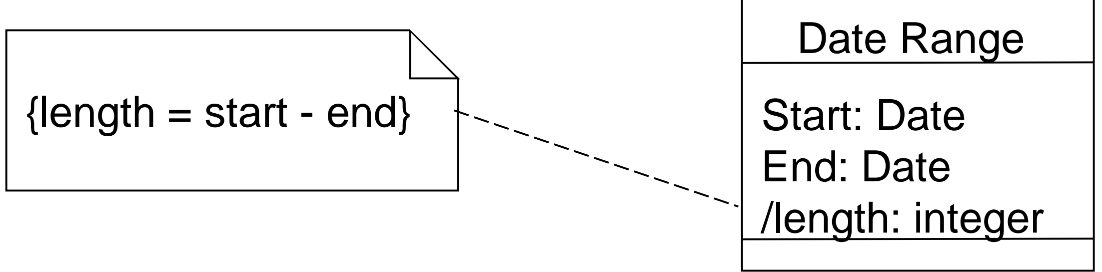


Annotation

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

Good Analysis Classes

- What makes a good analysis class?
  - Its name reflects its intent.
  - It is a crisp abstraction that models *one specific element of the problem domain*.
  - It maps to a clearly identifiable feature of the problem domain.
  - It has a small, well-defined set of *responsibilities*:
    - a responsibility is a contract or obligation that a class has to its clients;
    - a responsibility is a semantically cohesive set of operations;
    - there should only be about three to five responsibilities per class.
  - It has high cohesion – all features of the class should help to realize its intent.
  - It has low coupling – a class should only collaborate with a small number of other classes to realize its intent.

Bad Analysis Classes

- What makes a bad analysis class?
  - A functoid- a class with only one operation.
  - An omnipotent class -a class that does everything (classes with "system" or "controller" in their name *may* need closer scrutiny).
  - A class with a deep inheritance tree -in the real world inheritance trees tend to be shallow.
  - A class with low cohesion.
  - A class with high coupling.
  - Many very small classes in a model – merging should be considered.
  - Few but large classes in a model – decomposition should be considered.

Class Identification Techniques

- Noun/Verb Analysis (Grammatical Parsing)
- CRC Analysis
- Use-Case-Based Analysis
- Real-World Analysis

Noun/verb analysis (*Grammatical Parsing*)

1. Collect as much relevant information about the problem domain as possible; suitable sources of information are:
   - The requirements model
   - The use case model
   - The project glossary
   - Any other document (architecture, vision documents, etc.)

2. Analyze the documentation:
   - Look for **nouns or noun phrases** -these are candidate classes or attributes.
   - Look for **verbs or verb phrases** -these are candidate responsibilities or operations.
     - Always think about running methods on objects.

       e.g. given Number objects "x" and "y"

       x.add(y) is more OO than x = add(x, y)

3. Make a tentative allocation of the attributes and responsibilities to the classes.