



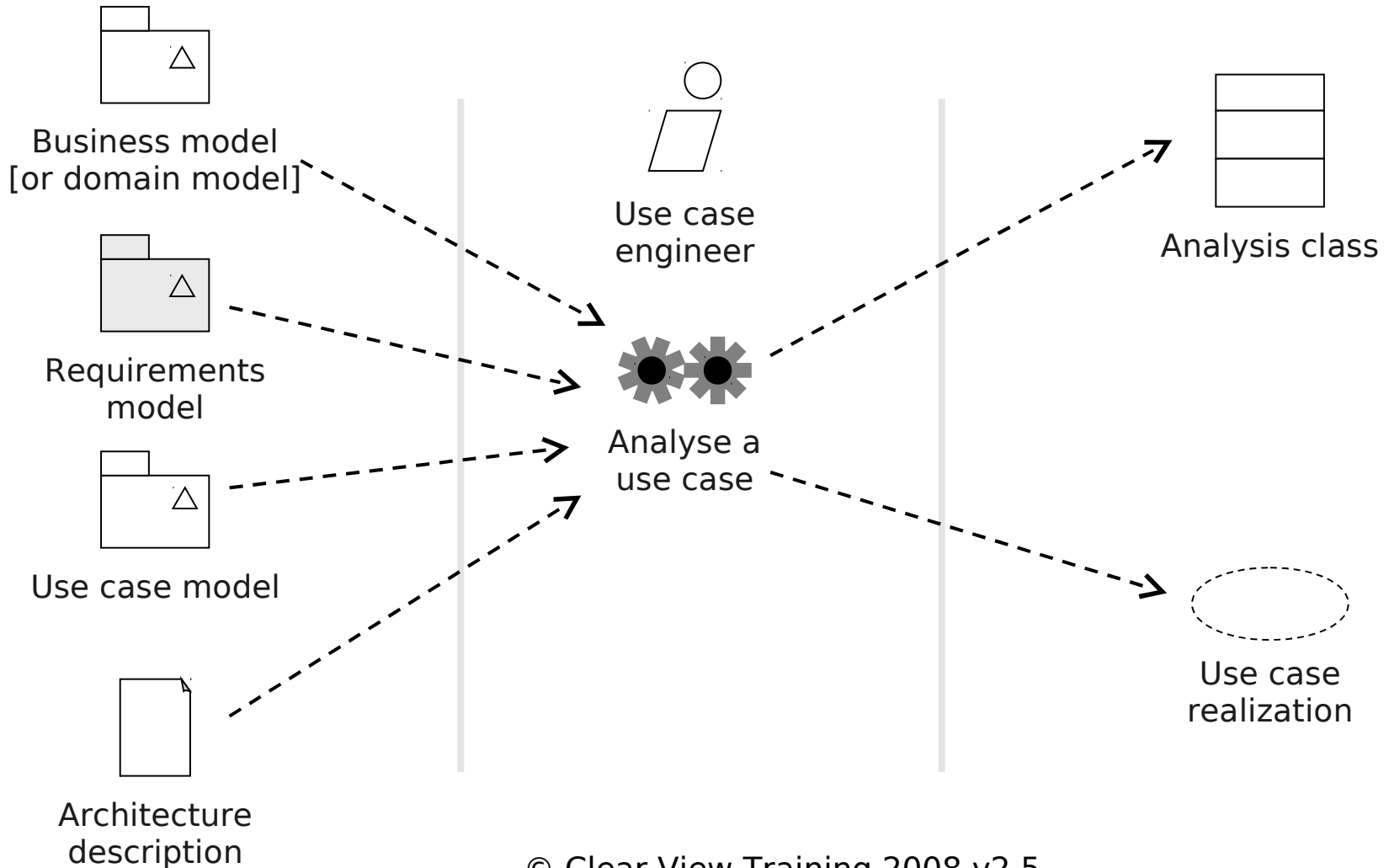
OO Analysis and Design with UML 2 and UP

Dr. Jim Arlow,
Zuhlke Engineering Limited

Analysis - finding analysis classes

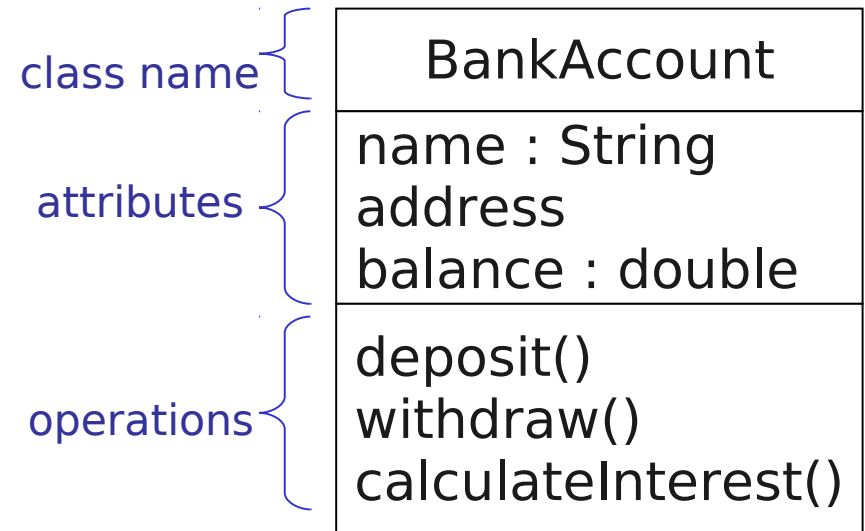


Analyse a use case



What are Analysis classes?

- Analysis classes represent a crisp abstraction in the problem domain
 - They may ultimately be refined into one or more design classes
- All classes in the Analysis model should be Analysis classes
- Analysis classes have:
 - A very “high level” set of attributes. They *indicate* the attributes that the design classes *might* have.
 - Operations that specify at a high level the key services that the class must offer. In Design, they will become actual, implementable, operations.
- Analysis classes must map onto real-world business concepts



We always specify attribute types if we know what they are!

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 onto a clearly identifiable feature of the problem domain
- It has *high cohesion*
 - Cohesion is the degree to which a class models a single abstraction
 - Cohesion is the degree to which the *responsibilities* of the class are semantically related
- It has *low coupling*
 - Coupling is the degree to which one class depends on others
- Rules of thumb:
 - 3 to 5 responsibilities per class
 - Each class collaborates with others
 - Beware many very small classes
 - Beware few but very large classes
 - Beware of “functoids”
 - Beware of “omnipotent” classes
 - Avoid deep inheritance trees

A *responsibility* is a contract or obligation of a class - it resolves into operations and attributes



Finding classes

- Perform noun/verb analysis on documents:
 - Nouns are candidate classes
 - Verbs are candidate *responsibilities*
- Perform CRC card analysis
 - A brainstorming technique using sticky notes
 - Useful for brainstorming, Joint Application Development (JAD) and Rapid Application development (RAD)
- With both techniques, beware of spurious classes:
 - Look for *synonyms* - different words that mean the same
 - Look for *homonyms* - the same word meaning different things
- Look for "hidden" classes!
 - Classes that don't appear as nouns or as cards

Noun/verb analysis procedure

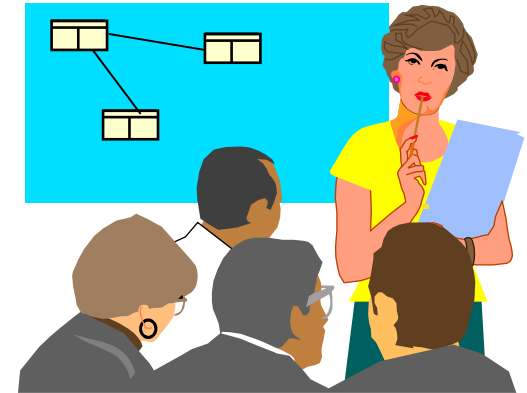
- Collect all of the relevant documentation
 - Requirements document
 - Use cases
 - Project Glossary
 - Anything else!
- Make a list of nouns and noun phrases
 - These are candidate classes or attributes
- Make a list of verbs and verb phrases
 - These are candidate responsibilities
- Tentatively assign attributes and responsibilities to classes

CRC card procedure

Class Name: BankAccount	
Responsibilities:	Collaborators:
Maintain balance	Bank

things
the class
does

things the
class works
with



- Class, Responsibilities and Collaborators
- Separate information collection from information analysis
 - Part 1: Brainstorm
 - All ideas are good ideas in CRC analysis
 - Never argue about something - write it down and analyse it later!
 - Part 2: Analyse information - consolidate with noun/verb



Other sources of classes

- Physical objects
- Paperwork, forms etc.
 - Be careful with this one – if the existing business process is very poor, then the paperwork that supports it might be irrelevant
- Known interfaces to the outside world
- Conceptual entities that form a cohesive abstraction e.g. LoyaltyProgramme



Summary

- We've looked at what constitutes a well-formed analysis class
- We have looked at two analysis techniques for finding analysis classes:
 - Noun verb analysis of use cases, requirements, glossary and other relevant documentation
 - CRC analysis



Analysis - relationships

What is a relationship?

- A *relationship* is a connection between modelling elements
- In this section we'll look at:
 - *Links* between objects
 - *Associations* between classes
 - aggregation
 - composition
 - association classes

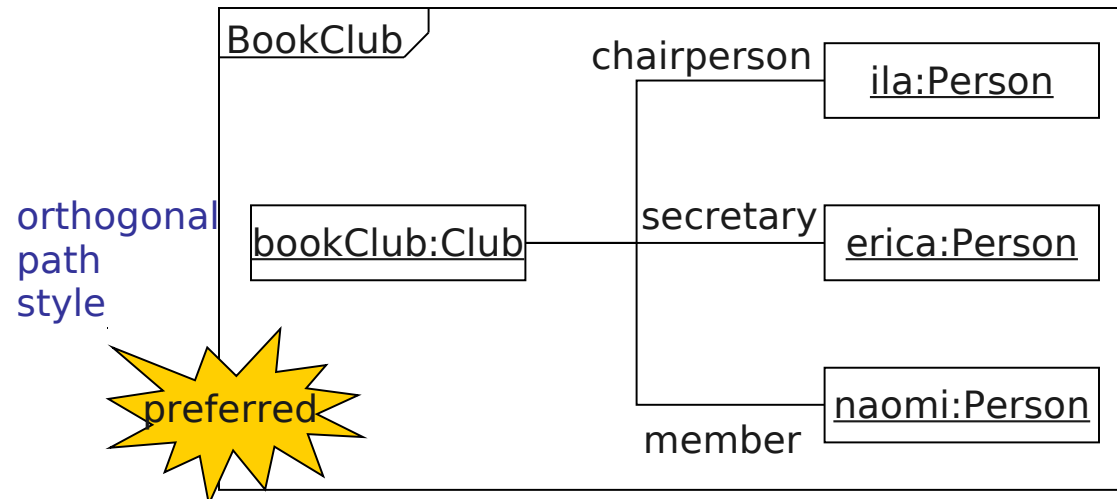
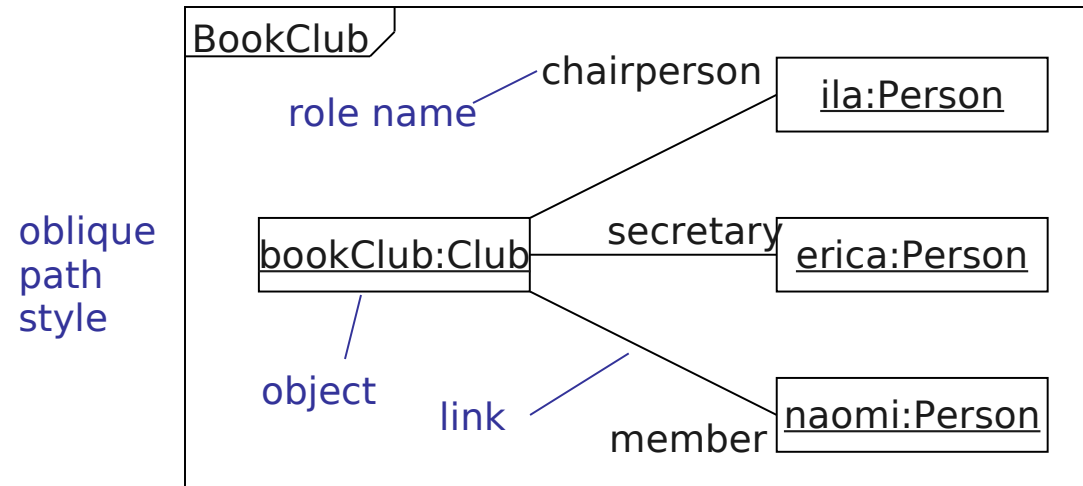


What is a link?

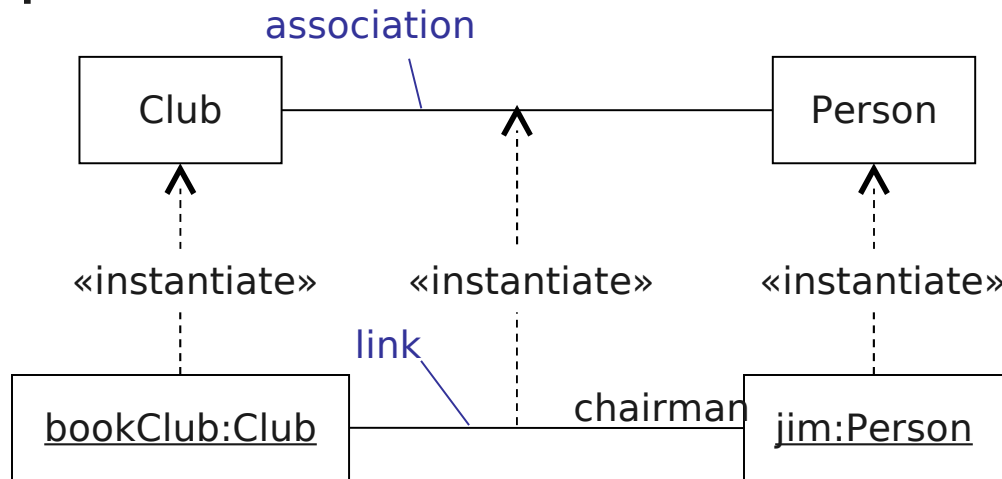
- Links are connections between objects
 - Think of a link as a telephone line connecting you and a friend. You can send messages back and forth using this link
- Links are the way that objects communicate
 - Objects send messages to each other via links
 - Messages invoke operations
- OO programming languages implement links as object references or pointers. These are unique handles that refer to specific objects
 - When an object has a reference to another object, we say that there is a *link* between the objects

Object diagrams

- Paths in UML diagrams (lines to you and me!) can be drawn as orthogonal, oblique or curved lines
- We can combine paths into a tree *if* each path has the same properties

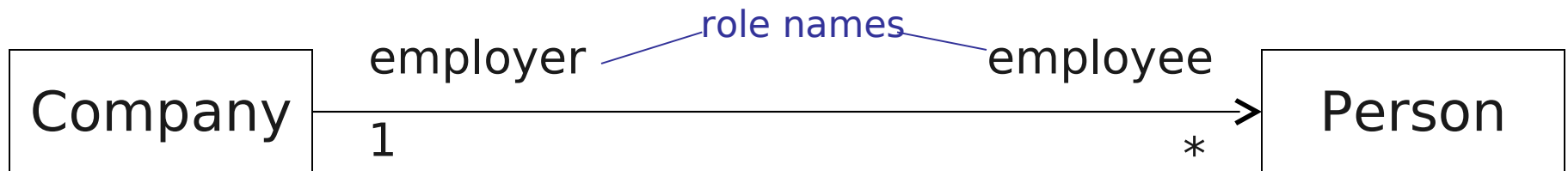


What is an association?



- Associations are relationships between classes
- Associations between classes indicate that there are links between objects of those classes
- A link is an instantiation of an association just as an object is an instantiation of a class

Association syntax

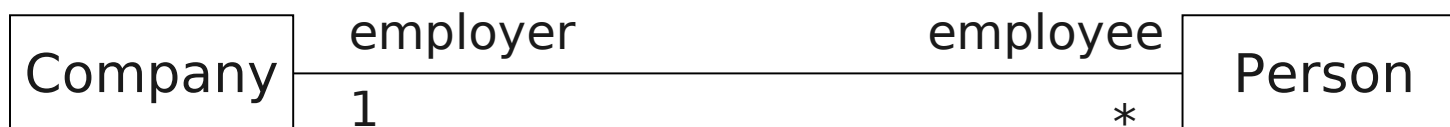


- An association can have role names *or* an association name
 - It's bad style to have both!
- The black triangle indicates the direction in which the association name is read:
 - "A Company employs many Persons"

Multiplicity



A Company employs many People



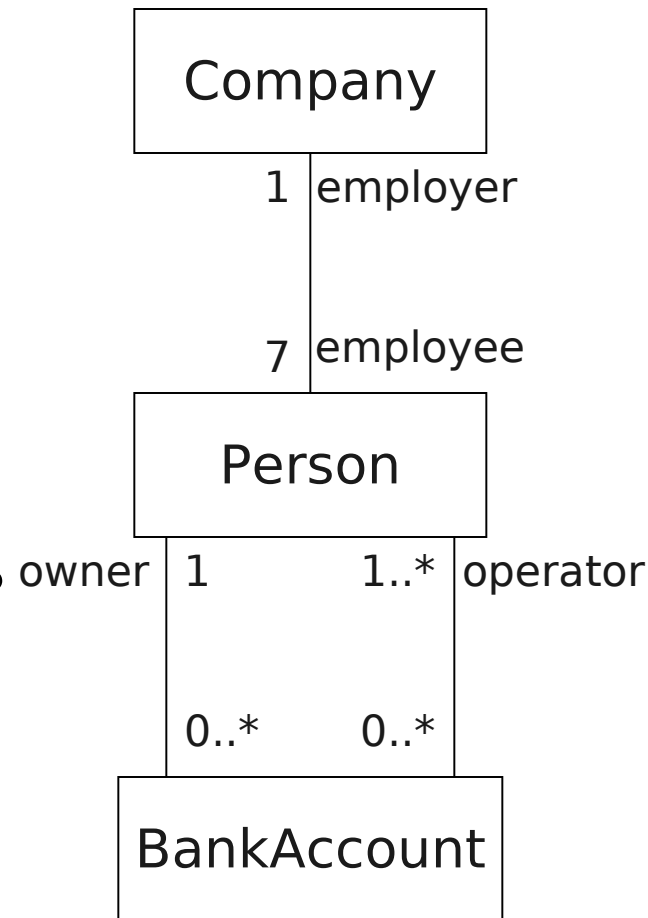
Each Person works for one Company

- Multiplicity is a constraint that specifies the number of objects that can participate in a relationship at *any point in time*
- If multiplicity is not explicitly stated in the model then it is undecided - *there is no default multiplicity*

multiplicity syntax: minimum..maximum	
0..1	zero or 1
1	exactly 1
0..*	zero or more
*	zero or more
1..*	1 or more
1..6	1 to 6

Multiplicity exercise

- How many
 - Employees can a Company have?
 - Employers can a Person have?
 - Owners can a BankAccount have?
 - Operators can a BankAccount have?
 - BankAccounts can a Person have?
 - BankAccounts can a Person operate?



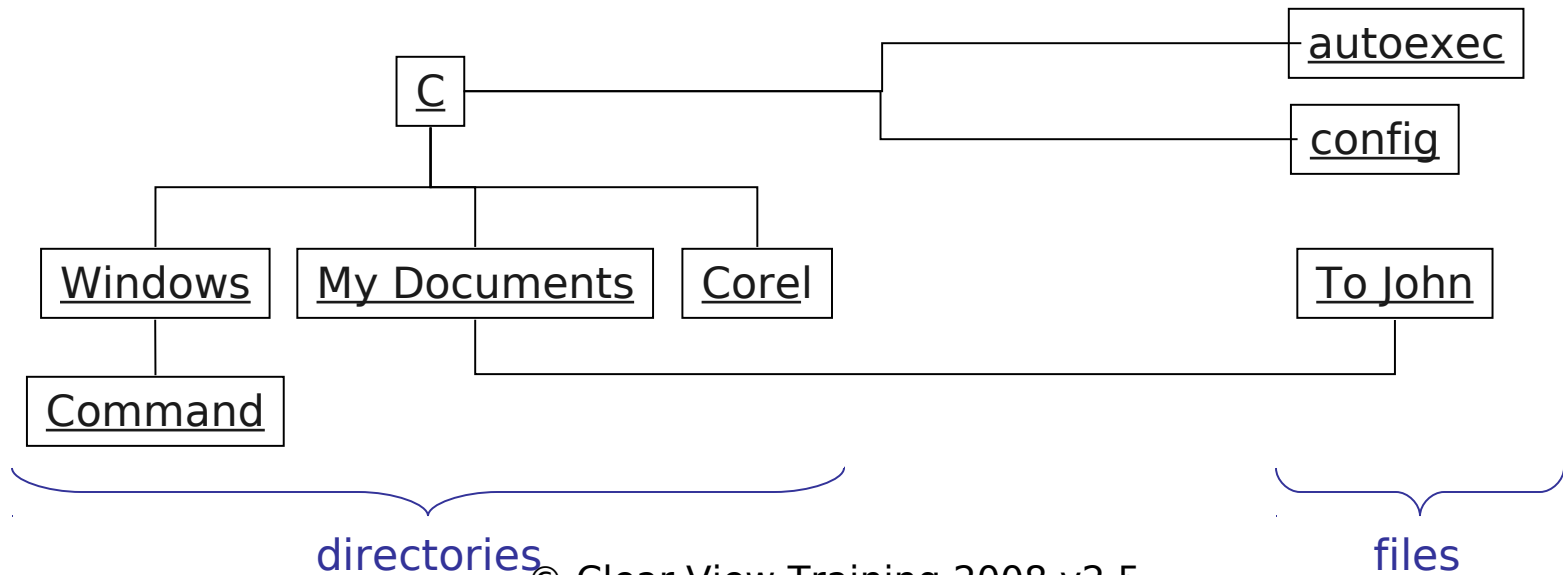
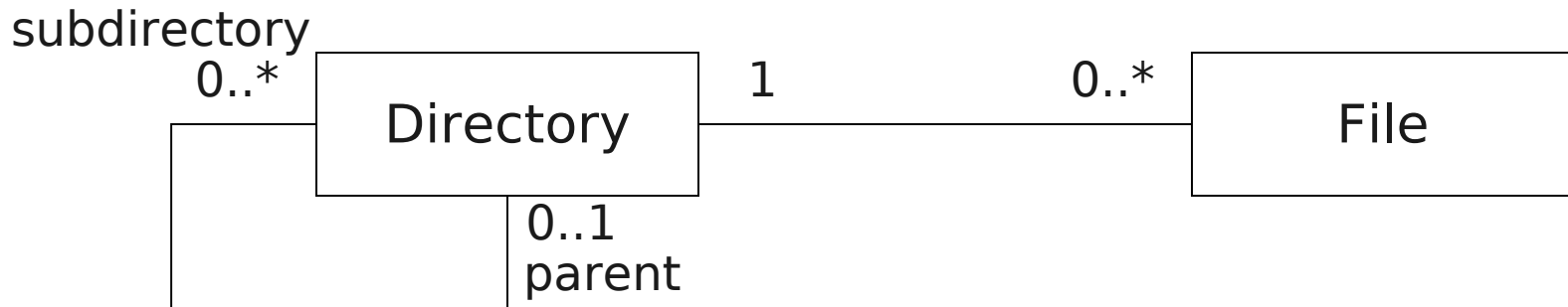
Exercise

- Model a computer file system. Here are the minimal facts you need:
 - The basic unit of storage is the file
 - Files live in directories
 - Directories can contain other directories
- Use your own knowledge of a specific file system (e.g. Windows 95 or UNIX) to build a model

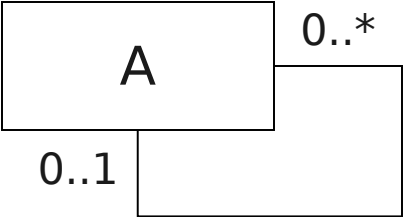
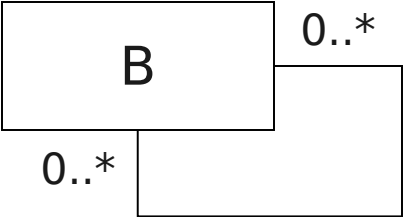
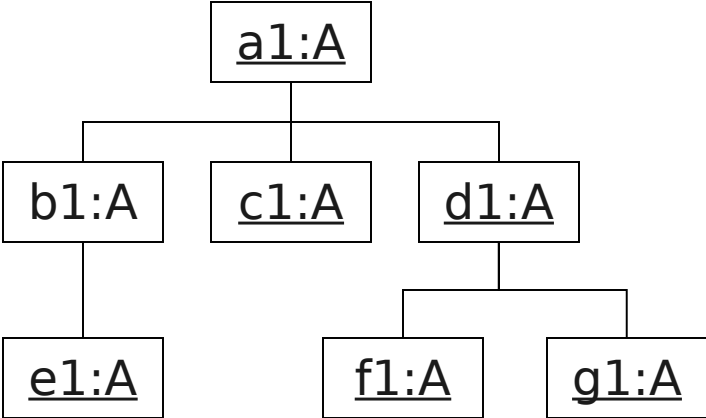
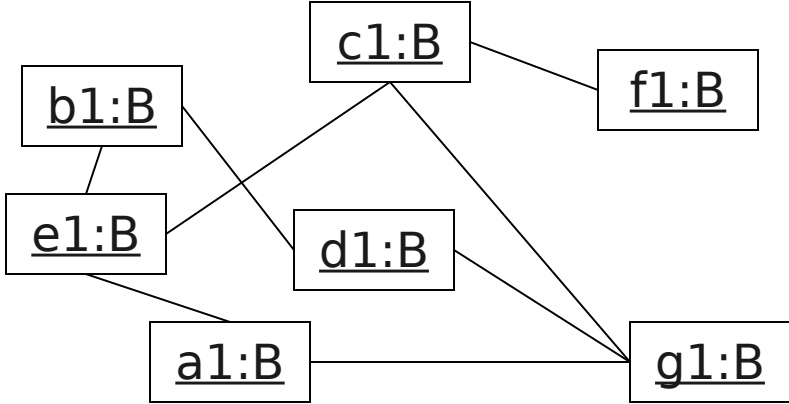


Hint: a class can have an association to itself!

Reflexive associations

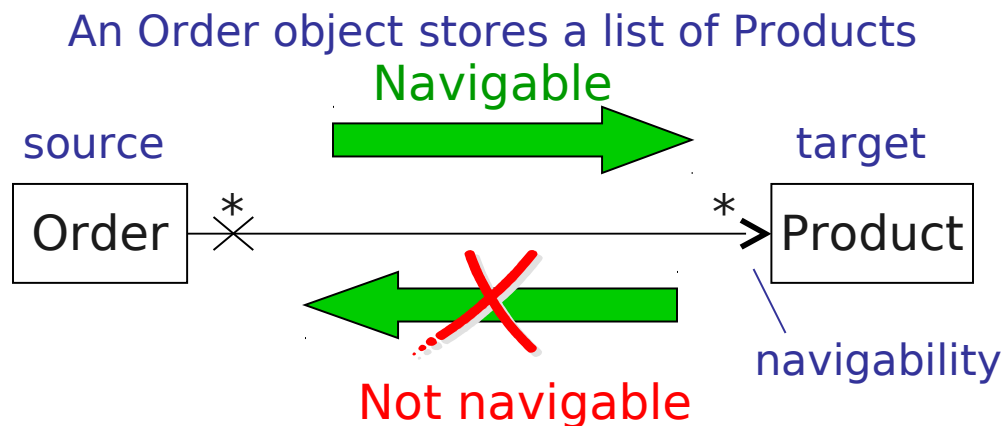


Hierarchies and networks

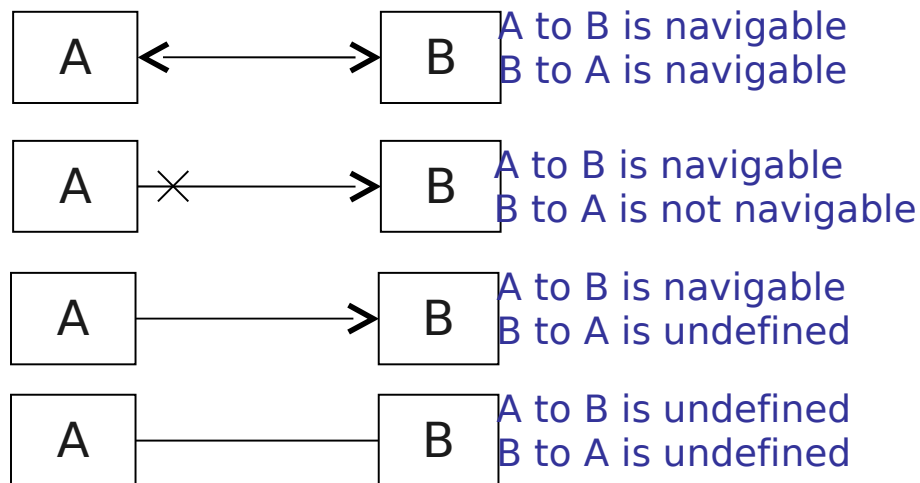
hierarchy	network
	
	
<p>in an association hierarchy, each object has <i>zero or one</i> object directly above it</p>	<p>in an association network, each object has zero or many objects directly above it</p>

Navigability

- Navigability indicates that it is possible to traverse from an object of the *source* class to objects of the *target* class
 - Objects of the source class may reference objects of the target class using the role name
- Even if there is *no* navigability it might still be possible to traverse the relationship via some indirect means. However the computational cost of the traversal might be very high



A Product object **does not** store a list of Orders

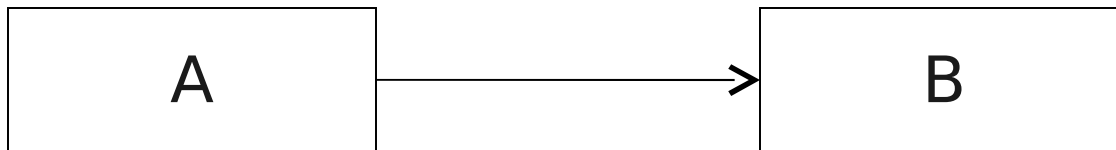


Navigability - standard practice

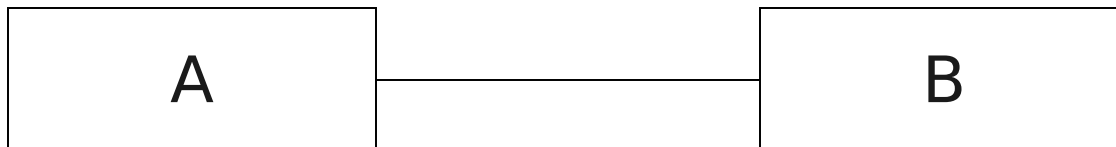
- Strict UML 2 navigability can clutter diagrams so the UML standard suggests three possible modeling idioms:
 - Show navigability explicitly on diagrams with crosses and arrows
 - Omit all navigability from diagrams
 - Omit crosses from diagrams

- bi-directional associations have no arrows
- unidirectional associations have a single arrow
- you can't show associations that are not navigable in either direction (not useful anyway!)

standard practice

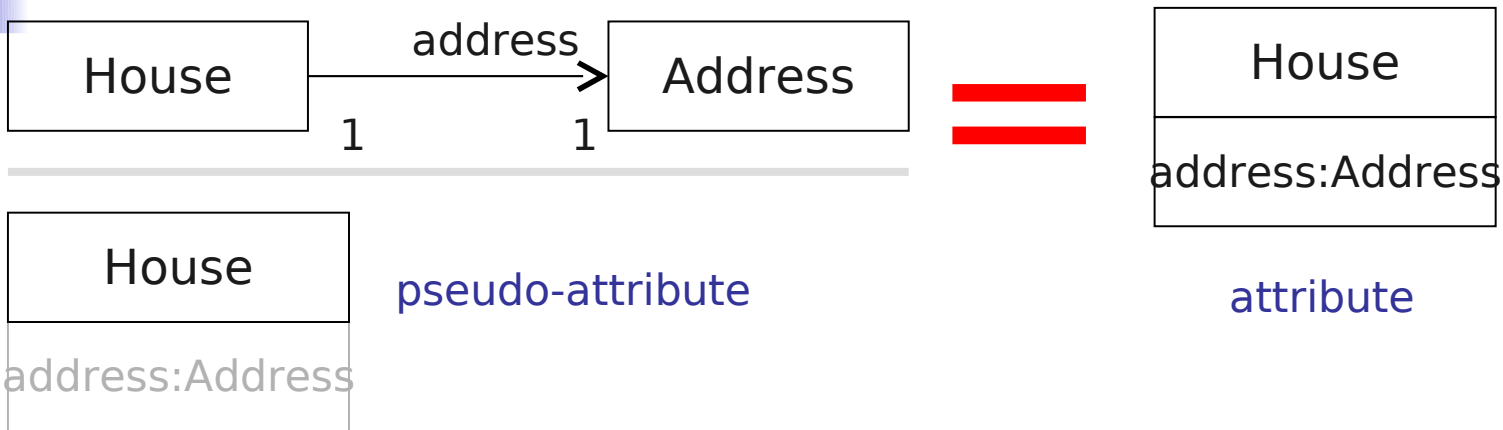


A to B is navigable
B to A is not navigable



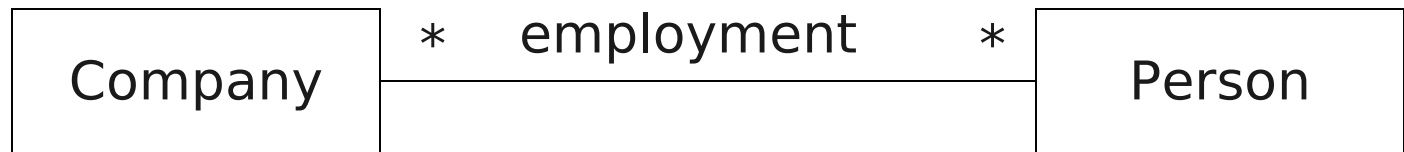
A to B is navigable
B to A is navigable

Associations and attributes



- If a navigable relationship has a role name, it is as though the source class has a pseudo-attribute whose attribute name is the role name and whose attribute type is the target class
- Objects of the source class can refer to objects of the target class using this pseudo-attribute
- Use associations when:
 - The target class is an important part of the model
 - The target class is a class that you have designed yourself and which must be shown on the model
- Use attributes when:
 - The target class is *not* an important part of the model e.g. a primitive type such as number, string etc.
 - The target class is just an implementation detail such as a bought-in component or a library component e.g. `Java.util.Vector` (from the Java standard libraries)

Association classes



Each Person object can work for many Company objects.

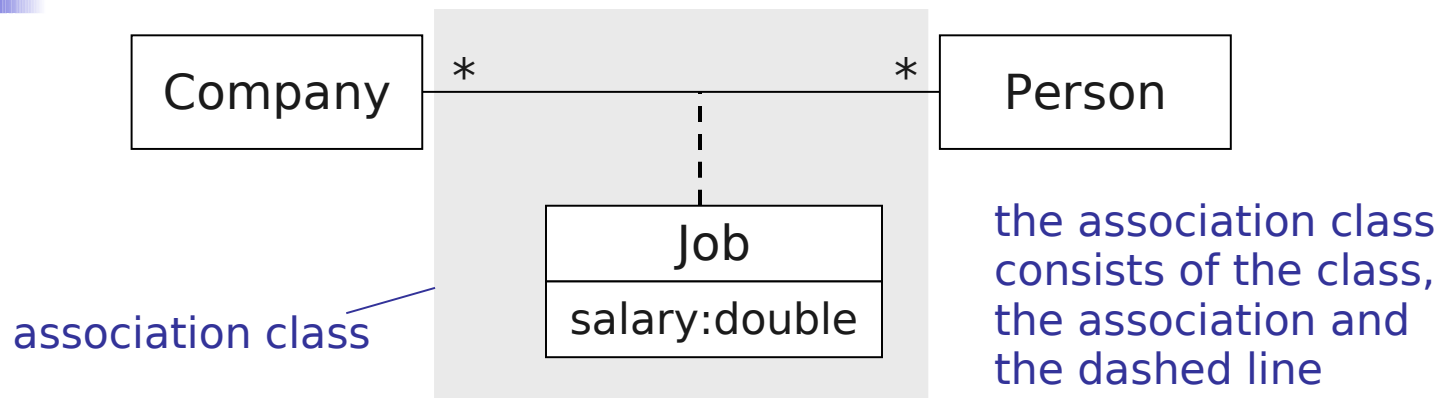
Each Company object can employ many Person objects.

When a Person object is employed by a Company object, the Person has a salary.

But where do we record the Person's salary?

- Not on the Person class - there is a different salary for each employment
- Not on the Company class - different Person objects have different salaries
- The salary is a property of the employment relationship itself
 - every time a Person object is employed by a Company object, there is a salary

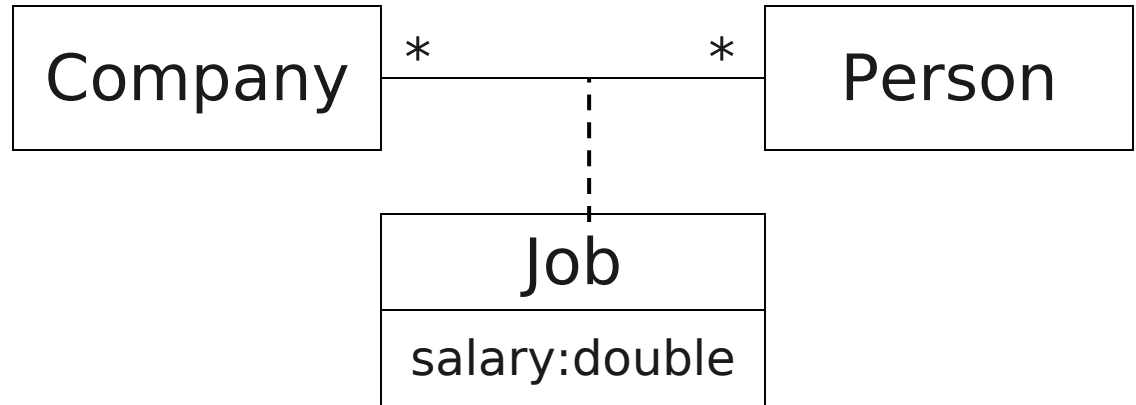
Association class syntax



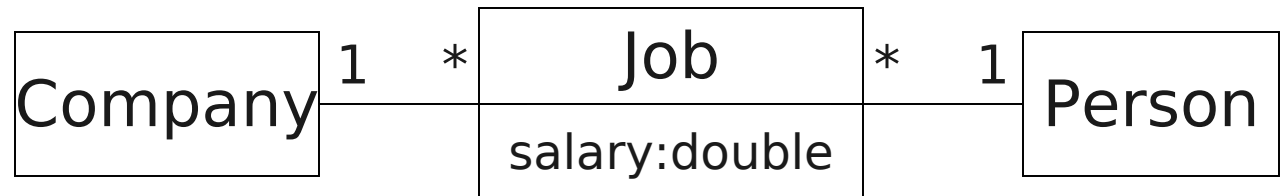
- We model the association itself as an association class. One instance of this class exists for each link between a **Person** object and a **Company** object
 - Instances of the association class are links that have attributes and operations
 - Can only use association classes when there is *one unique link* between two specific objects. This is because the identity of links is determined exclusively by the identities of the objects on the ends of the link
- We can place the salary and any other attributes or operations which are really features of the association into this class

Using association classes

If we use an association class, then a particular Person can have only *one* Job with a particular Company



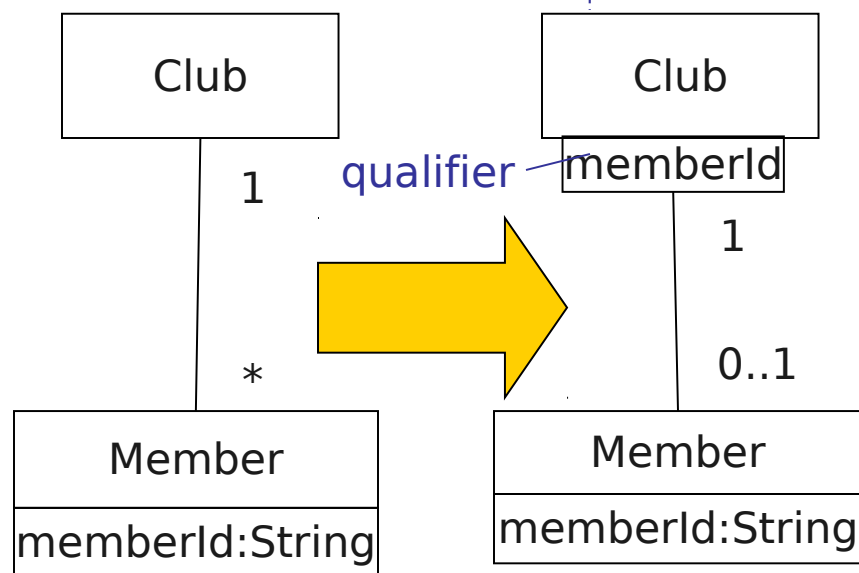
If, however a particular Person can have *multiple* jobs with the same Company, then we must use a *reified association*



Qualified associations

- Qualified associations reduce an n to many association to an n to 1 association by specifying a unique object (or group of objects) from the set
- They are useful to show how we can look up or navigate to specific objects
- Qualifiers usually refer to an attribute on the target class

the combination (Club, memberId) specifies a unique target object





Summary

- In this section we have looked at:
 - Links – relationships between objects
 - Associations – relationships between classes
 - role names
 - multiplicity
 - navigability
 - association classes
 - qualified associations