Object-Oriented Design

Lecturer: Raman Ramsin

Lecture 17:

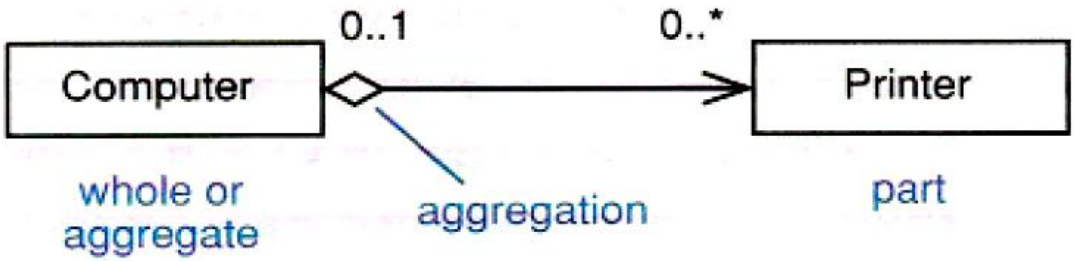
Refining Analysis Relationships

Refining Analysis Relationships

- Relationships in analysis are converted to implementable design relationships.
- Refining analysis relationships to design relationships involves:
  - adding navigability;
  - adding multiplicity to both ends of the association;
  - adding a role name at both ends of the association, or at at least on the target end of the association;
  - implementing one-to-one, one-to-many, many-to-one, and many-to-many associations;
  - implementing bidirectional associations and association classes;
  - using structured classifiers for modeling composition.

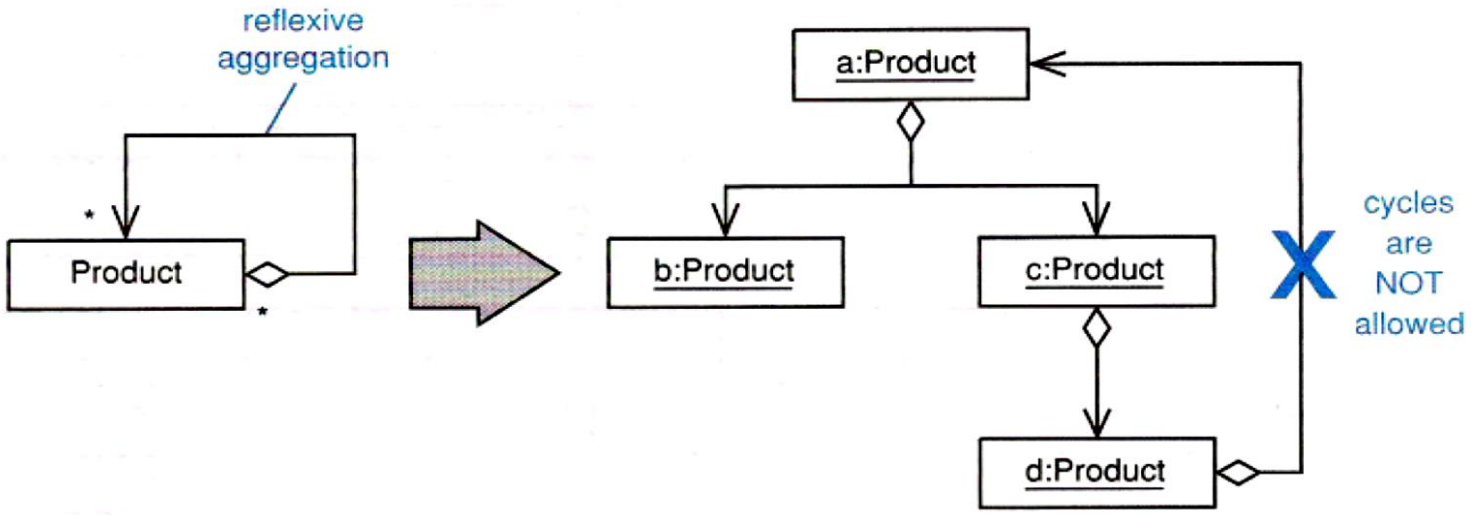
Aggregation Relationship

- Whole-part relationship where objects of one class act as the whole or *aggregate*, and objects of the other class act as the parts;
  - general semantics: Assembly, Containment, or Membership; can be used in analysis as well as design;
  - the whole uses the services of the parts; the parts service the requests of the whole;
  - the whole is the dominant, controlling side of the relationship; the part tends to be more passive;
  - aggregation is transitive: If C is part of B and B is part of A, then C is part of A.



Aggregation Relationship: Asymmetry

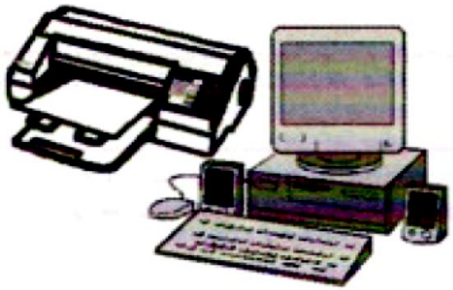
- Aggregation relationship is asymmetric:
  - a whole can never directly or indirectly be a part of itself;
  - there must never be a cycle in the aggregation graph.



Aggregation and Composition

- There are two types of aggregation relationship:
  - **Aggregation**;
  - **Composition Aggregation** - usually referred to simply as **Composition**.

Aggregation



Some objects are weakly related, like a computer and its peripherals

Composition

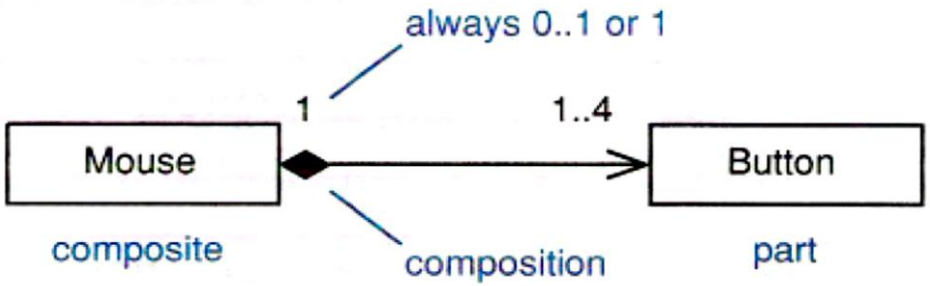
Some objects are strongly related, like a tree and its leaves

Aggregation: Semantics

- "Aggregation" is a weak Whole-Part relationship (like a computer system and its peripherals);
- The aggregate can sometimes exist independently of the parts, sometimes not;
- The parts may exist independently of the aggregate;
- It is possible to have shared ownership of the parts by several aggregates;
- Aggregation hierarchies and aggregation networks are possible;
- The whole always knows about the parts, but if the relationship is one-way from the whole to the part (which is typically the case), the parts don't know about the whole.

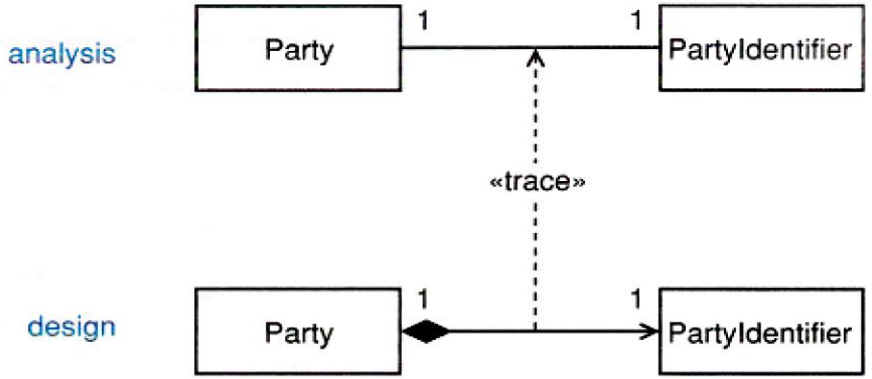
Composition

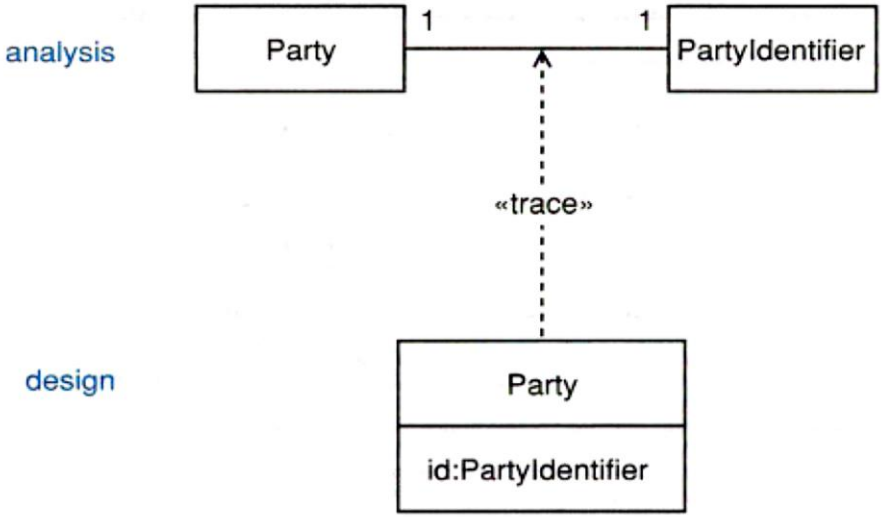
- A strong form of aggregation (like a tree and its leaves):
  - the parts belong to exactly one *composite* at a time;
  - the composite has sole responsibility for the disposition of all its parts - this means responsibility for their creation and destruction;
  - the composite may also release parts, provided responsibility for them is assumed by another object;
  - if the composite is destroyed, it must destroy all its parts or give responsibility for them over to some other object;
  - each part belongs to exactly one composite so you can only have composition hierarchies - composition networks are impossible.



Refining Analysis Relationships: One-to-One Association

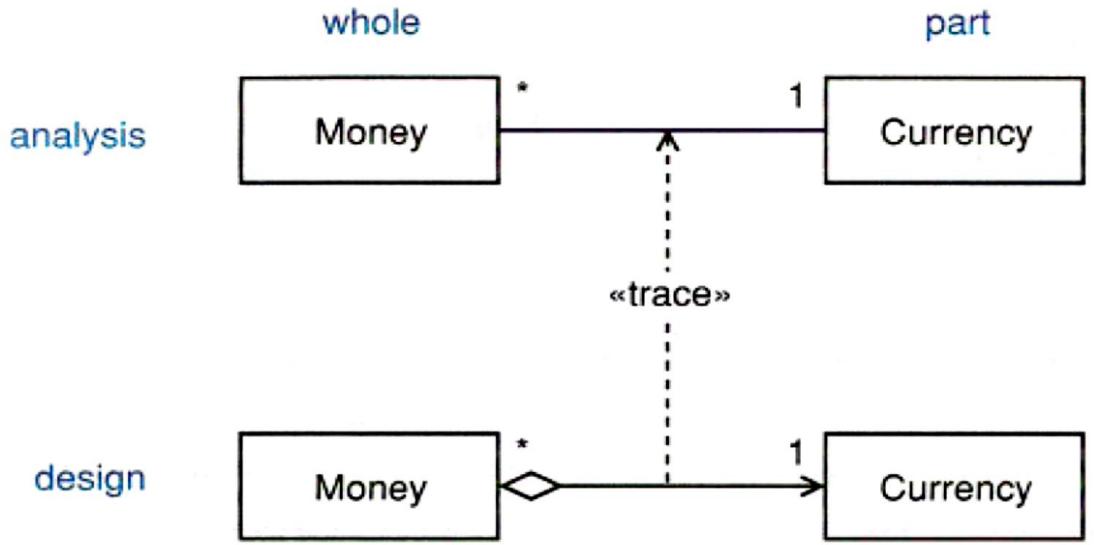
- Add navigability to the model; refine into Composition only if the semantics apply; you may also choose to merge the two classes.





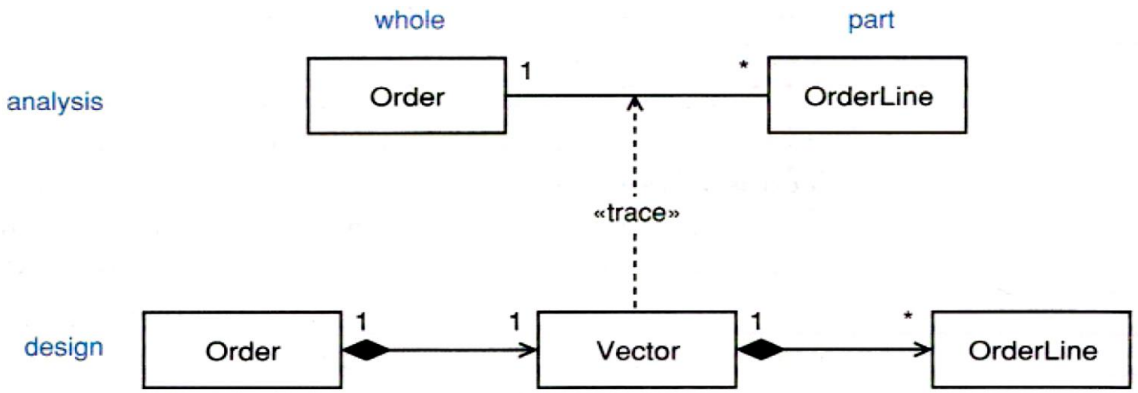
Refining Analysis Relationships: Many-to-One Association

- Add navigability; refine into Aggregation only if the semantics apply.



Refining Analysis Relationships: One-to-Many Association

- There is a collection of objects on the target side:
  - Use an inbuilt array (most OO languages directly support arrays) - they are generally quite inflexible but are usually fast.
  - Use a collection class - they are more flexible than inbuilt arrays and are faster than arrays when searching the collection is required (otherwise they are slower). A *Map* is frequently used.

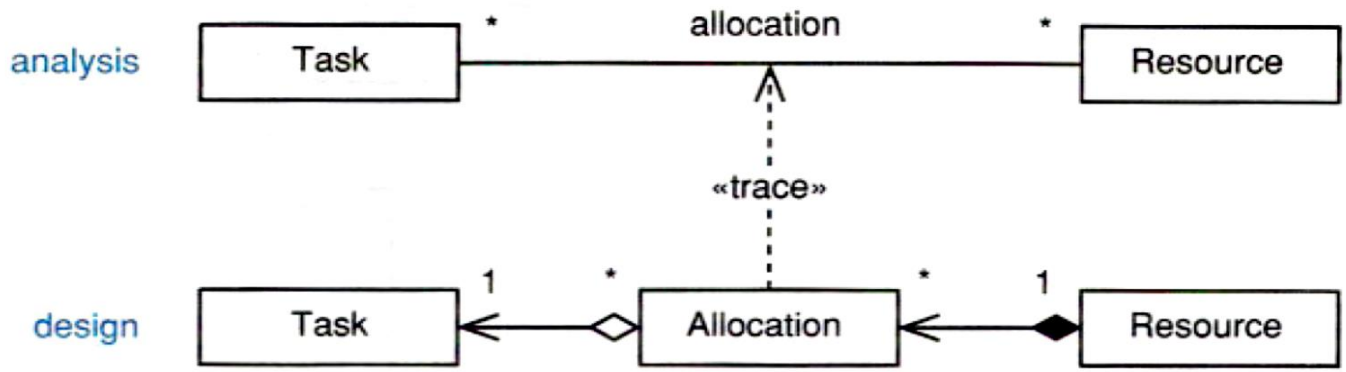


Reifying Analysis Relationships

- Some relationships are pure analysis artifacts and can be made implementable by the process of reification:
  1. Many-to-many associations
  2. Bidirectional associations
  3. Association classes

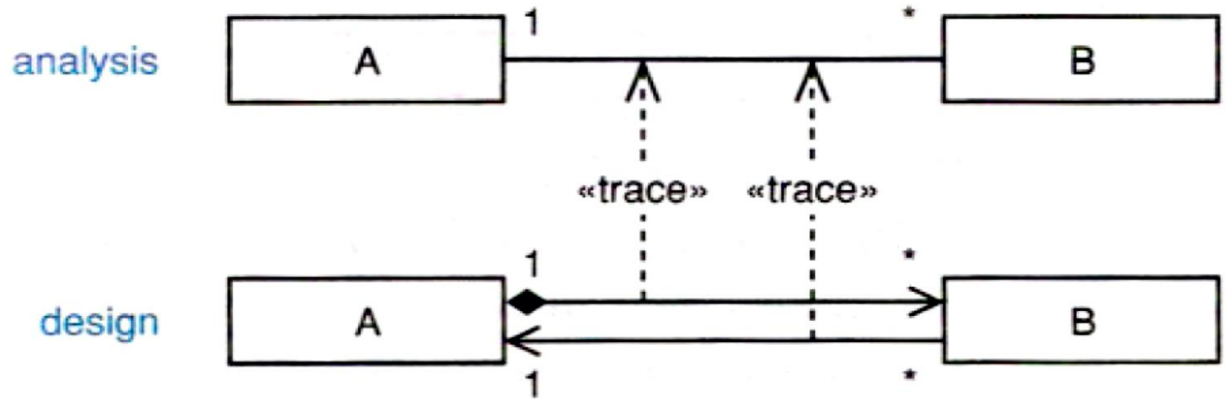
Reifying Relationships: Many-to-Many Associations

1. Add navigability; Refine into Aggregation or Composition only if the semantics apply.
2. Use a collection or reify the relationship into a class.



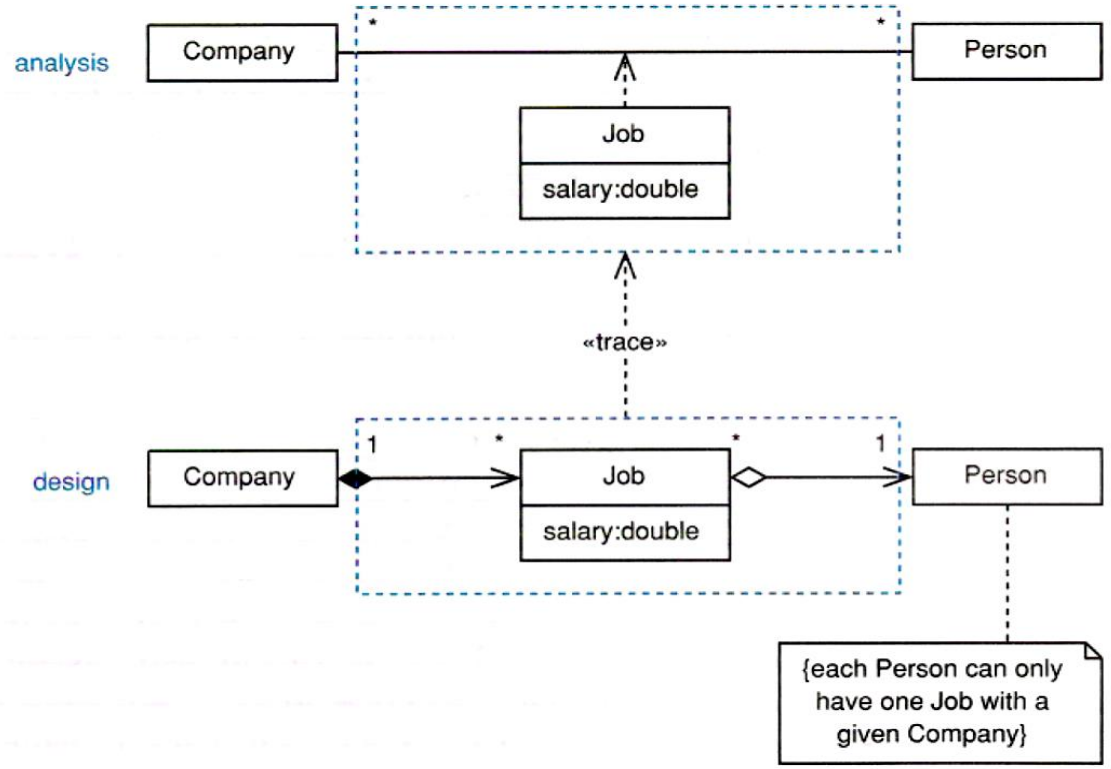
Reifying Relationships: Bidirectional Associations

- Replace with two unidirectional associations or a reified class or a bidirectional map; refine into Aggregation or Composition only if the semantics apply.



Reifying Relationships: Association Classes

1. Replace with a class (usually with the same name as the association class);
2. Add a constraint in a note to indicate that objects on each end of the reified relationship must form a unique pair.



Reference

- Arlow, J., Neustadt, I., *UML 2 and the Unified Process: Practical Object-Oriented Analysis and Design*, 2nd Ed. Addison-Wesley, 2005.