MACIASZEK, L.A. (2007):
***Requirements Analysis and System Design*, 3rd ed.**
Addison Wesley, Harlow England
ISBN 978-0-321-44036-5

Chapter 4
Moving from Analysis to Design

Topics

- Advanced class modeling
- Advanced generalization and inheritance modeling
- Advanced aggregation and delegation modeling
- Advanced interaction modeling

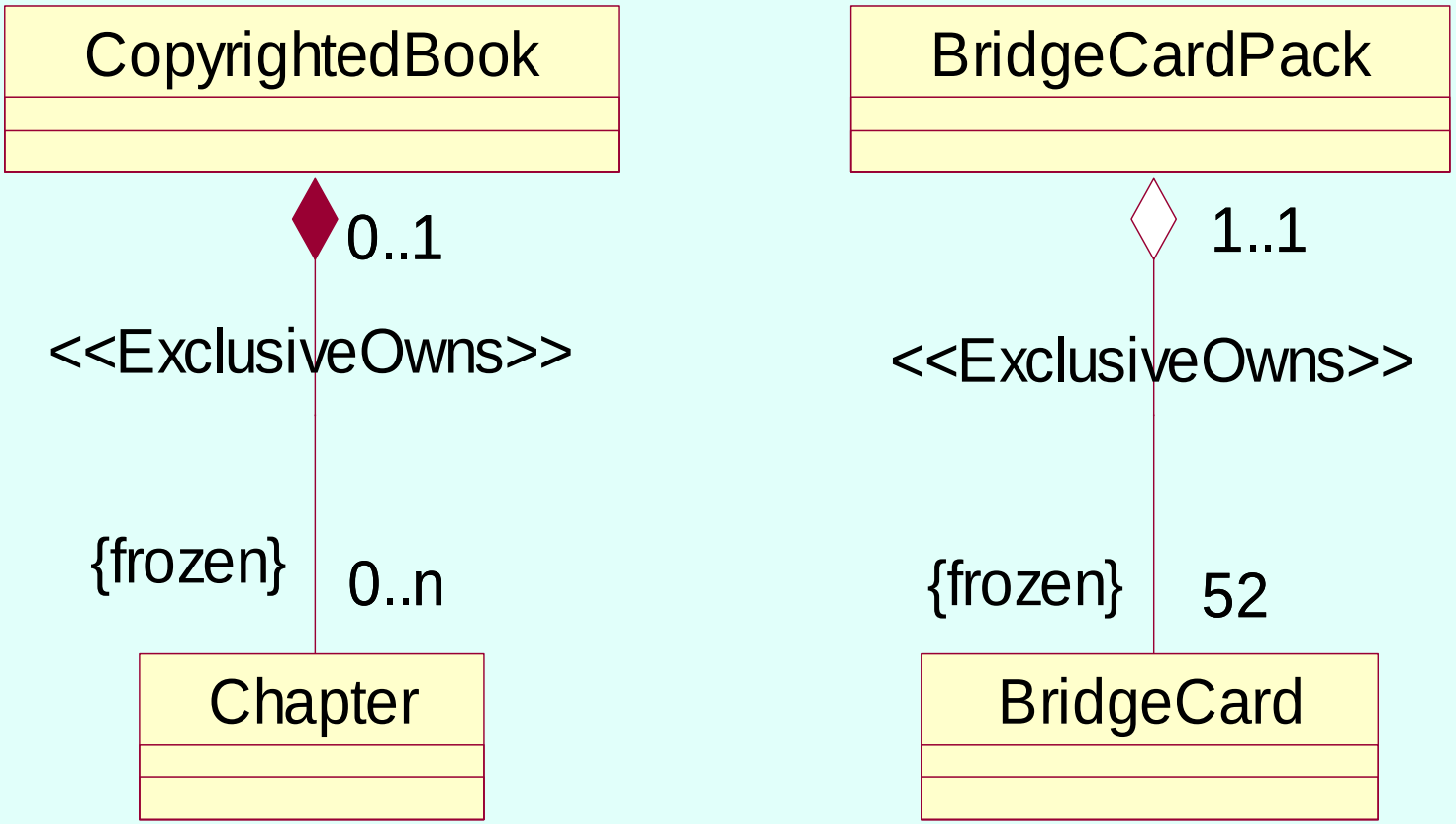
3. Advanced aggregation and delegation modeling

- *Aggregation* is the most powerful technique for managing the complexity of large systems through the allocation of classes to hierarchical layers of abstraction

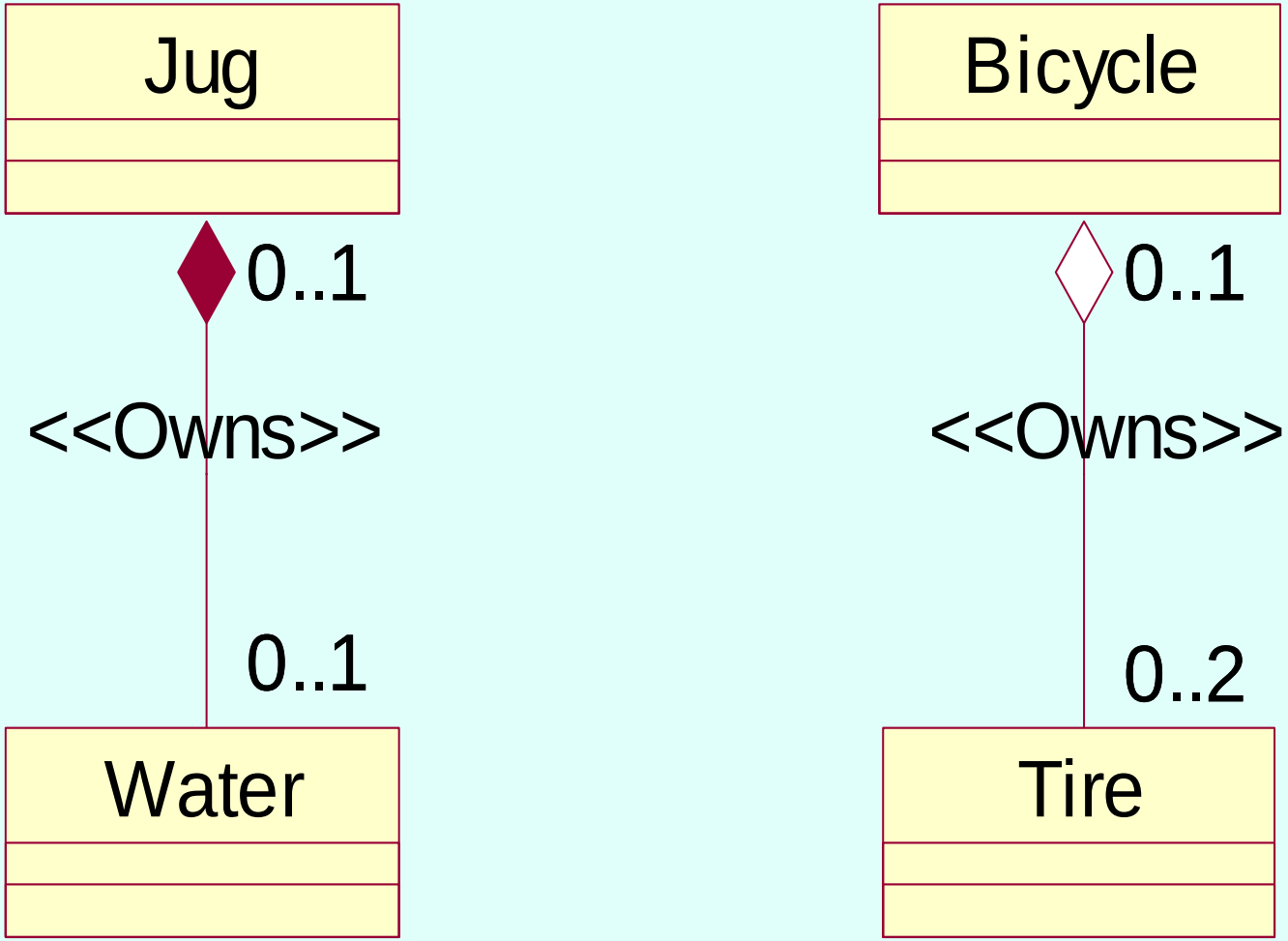
Putting more semantics into aggregation

- *Aggregation* (and its stronger variation – *composition*) is a containment relationship
  - A *composite class* contains one or more *component classes*
- In programming environments, aggregation is implemented in the same way as conventional associations
- More semantics needed:
  - "ExclusiveOwns" aggregation
  - "Owns" aggregation
  - "Has" aggregation
  - "Member" aggregation

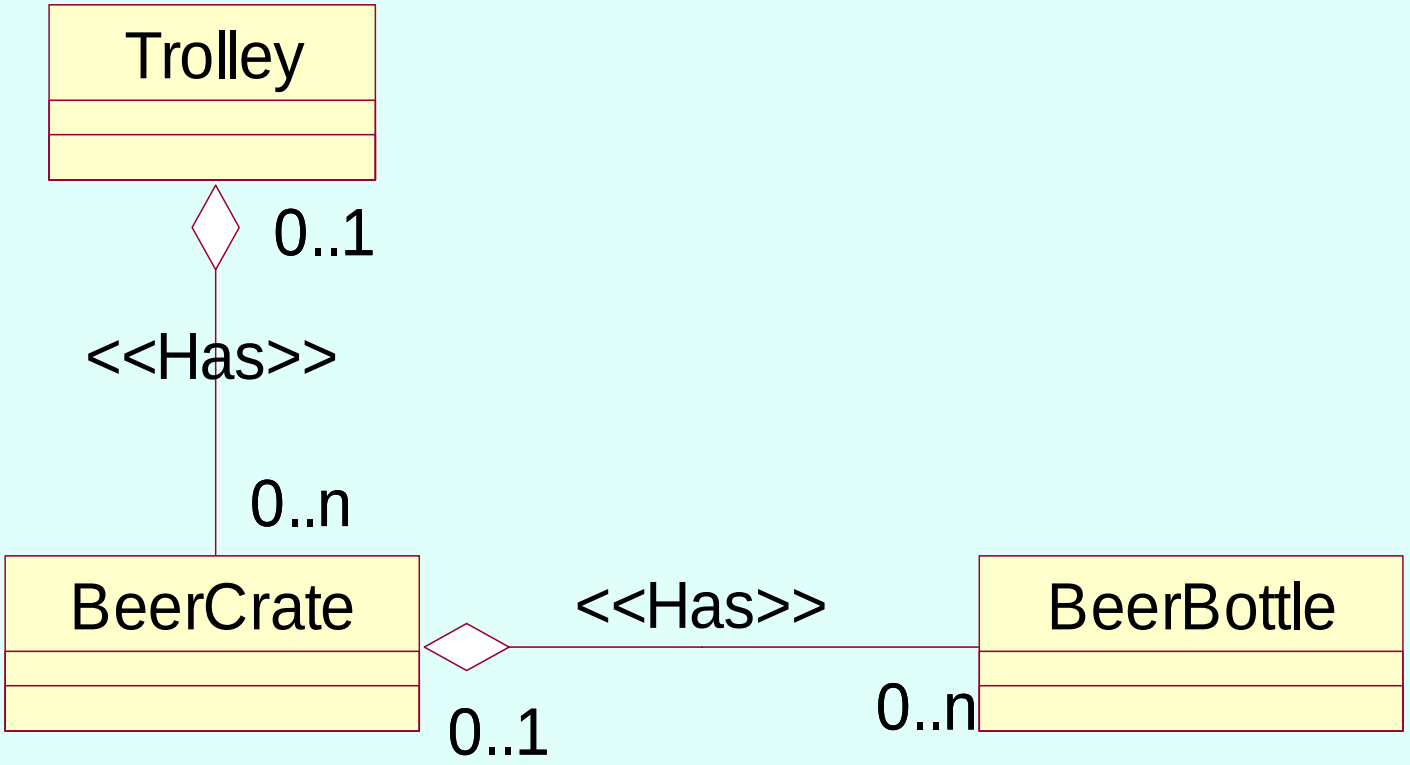
ExclusiveOwns aggregation



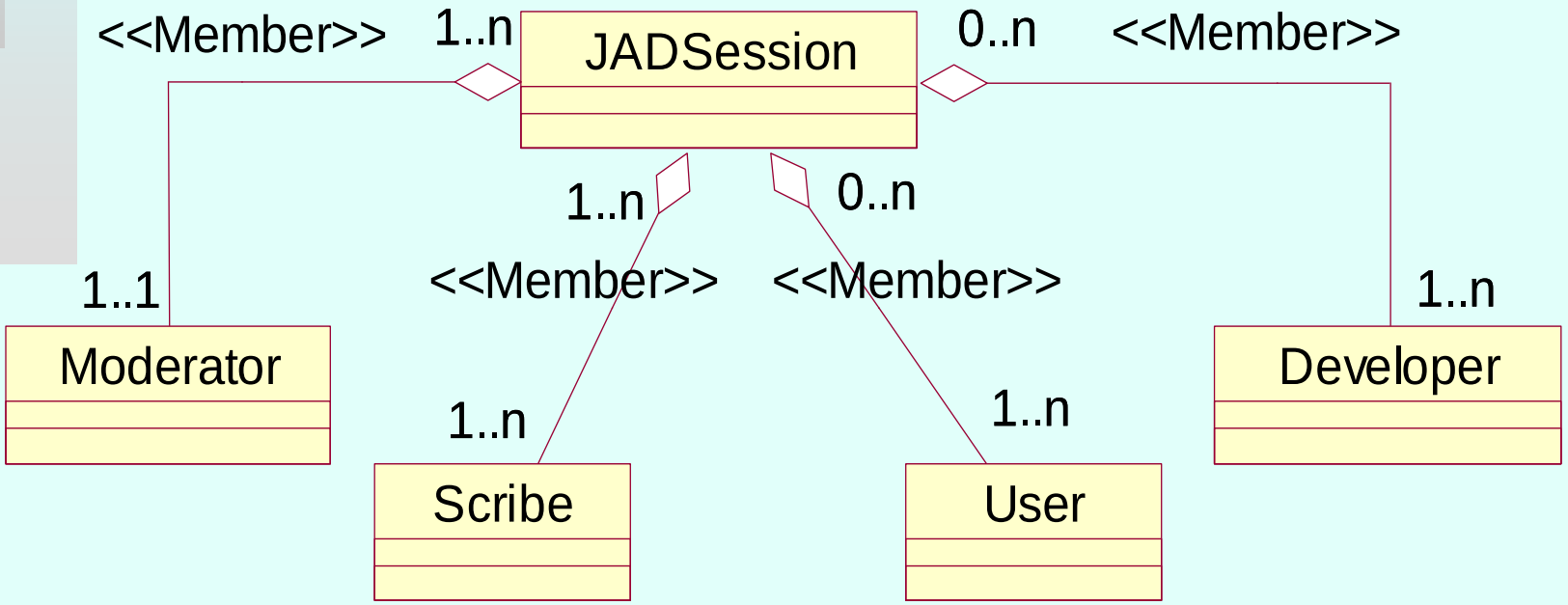
Owns aggregation



Has aggregation



Member aggregation



Delegation and prototypical systems

- The computational model of inheritance is based on the notion of a *class*
- However, it is possible to base the computational model on the notion of an *object*:
  - it uses aggregation
  - it is referred to as delegation – when an *outer object* cannot complete a task, it can call on the methods in one of its component objects (*inner objects*)
  - the functionality of the system is implemented by including (*cloning*) the functionality of existing objects in the newly required functionality
    - the existing objects are treated as prototypes for the creation of new objects
  - the inner object's interfaces may or may not be visible to objects other than the outer object

Delegation versus inheritance

- A delegation can model the inheritance, and vice versa → the same system functionality can be delivered with inheritance or with delegation → Treaty of Orlando
- From the *reuse* point of view, delegation comes very close to inheritance → an outer object reuses the implementation of the inner object
  - In inheritance, control is always returned to the originating object after the service has been accomplished
    - *self-recursion* always happens
    - sharing and reuse is normally determined statically → *anticipatory sharing*
  - In delegation, once control has been passed from an outer to an inner object, it stays there
    - *self-recursion* has to be explicitly planned
    - sharing and reuse is determined dynamically → *unanticipatory sharing*

Aggregation and holons

- Natural (living) systems are ‘holonic’ systems
  - ***Holon*** – an object that is both a part and a whole
  - Holons are hierarchically layered according to complexity → ***holarchies***
- Successful systems are arranged in holarchies that hide complexity in successively lower layers while providing greater levels of abstraction within the higher layers of their structures → this concept matches the semantics of ***aggregation***

Review Quiz 4.3

1. How is aggregation implemented in typical programming environments?
2. Which kind of aggregation needs to be specified with the “frozen” constraint?
3. What does aggregation use to reuse the implementation of component objects?