Systems Modeling

Lecture/Lab
2009/03/02

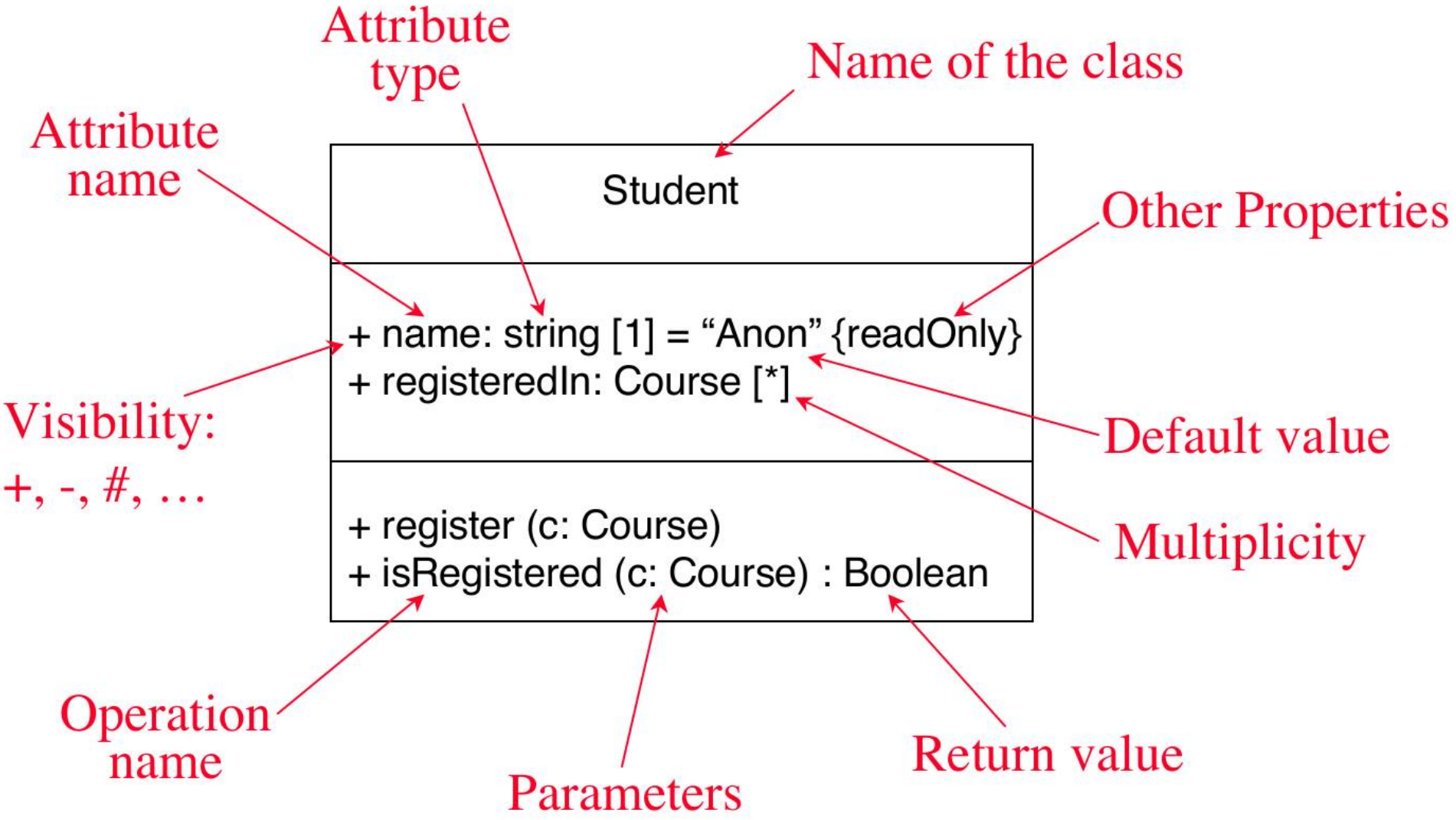
Distinguishing Objects (revisited) How to group them?

- How can these be distinguished?
  - all persons in the world for the purpose of sending mail
  - all persons in the world for the purpose of criminal investigations
  - all customers with safe deposit boxes in a given bank
  - all customers of a telephone company for billing purposes
  - all employees of a company to restrict access for security reasons

Class

- Set theory: Collection of sets or other mathematical objects that can be unambiguously defined by a property that all its members share
- Programming language construct
- Blueprint to create objects
- Groups objects
- Template for actual, in-memory, instances
- Includes attributes and methods that derived objects share

Anatomy



Taken from Steve Easterbrook, University of Toronto

Associations

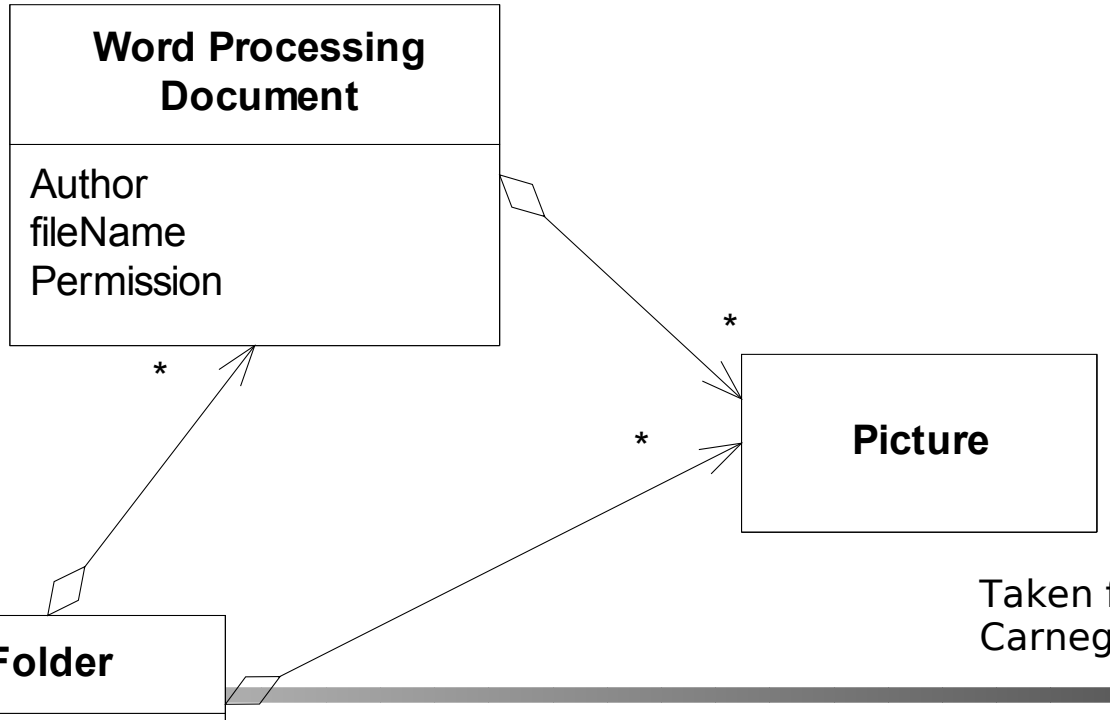
- Objects on both sides of the association can find each other
- The relation is consistent in time (unless removed)

Multiplicity
Indicates cardinality
- – 1:1 default
- 3 – exactly 3 object
- * (or n) - unbounded
- 1..* - 1 to eternity
- 3..9 – 3 to 9

Taken from Eran Toch, Carnegie Mellon University

Aggregation

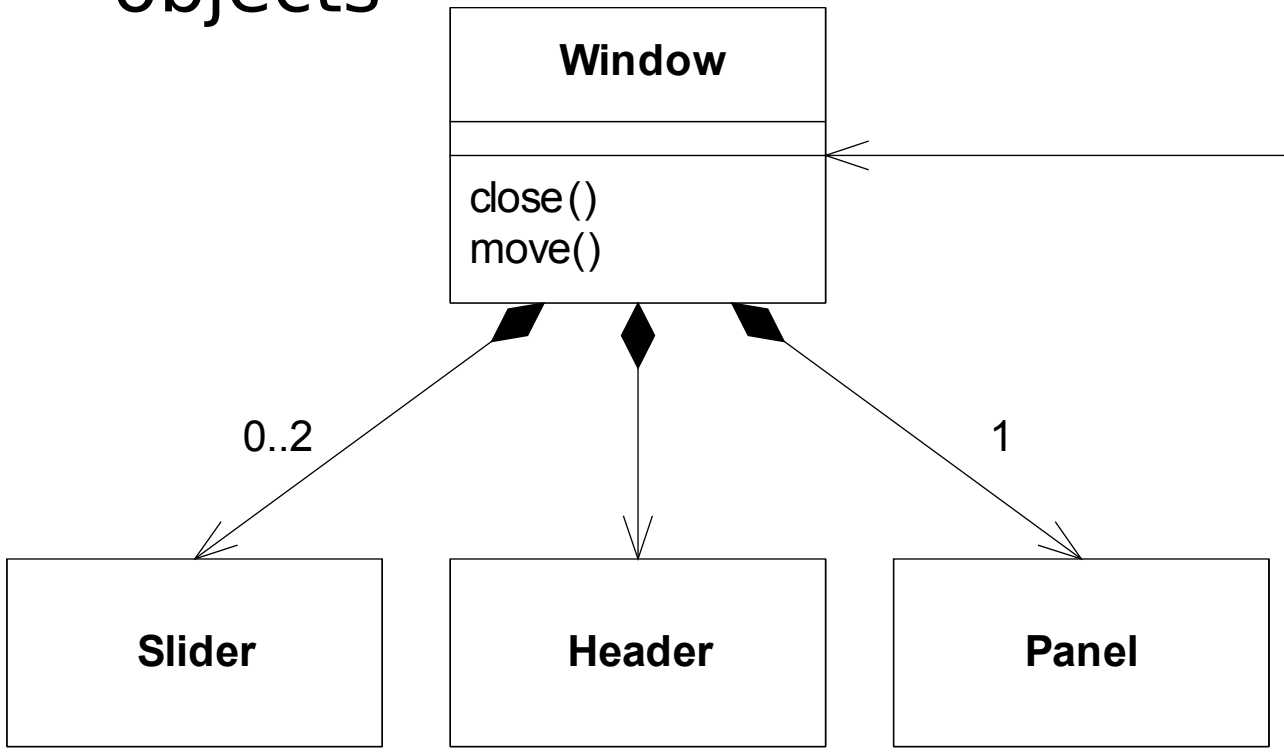
- "Whole-part" relationship between classes
- Assemble a class from other classes
  - Combined with "many" - assemble a class from a couple of instances of that class

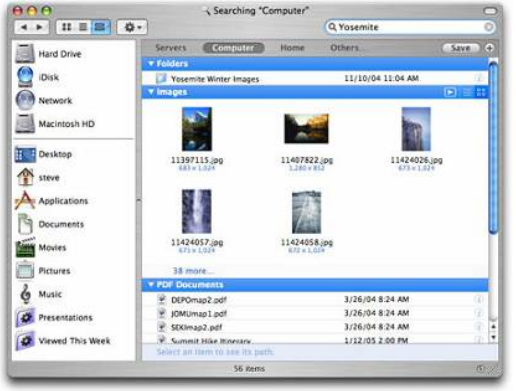


Taken from Eran Toch, Carnegie Mellon University

Composition

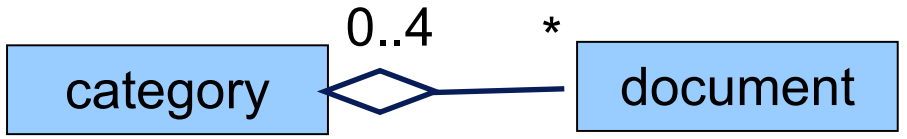
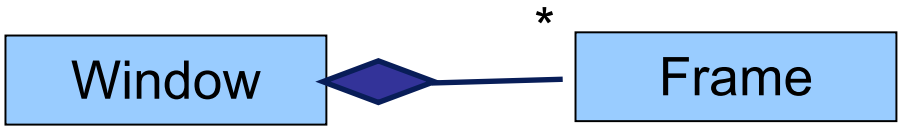
- Composition is a stronger form of aggregation
- Contained objects that live and die with the container
- Container creates and destroys the contained objects





Taken from Eran Toch, Carnegie Mellon University

Composition vs. Aggregation

Aggregation	Composition
Part can be shared by several wholes 	Part is always a part of a single whole 
Parts can live independently (i.e., whole cardinality can be 0..*)	Parts exist only as part of the whole. When the whole is destroyed, the parts are destroyed
Whole is not solely responsible for the object	Whole is responsible and should create/destroy the objects

Taken from Eran Toch,
Carnegie Mellon University

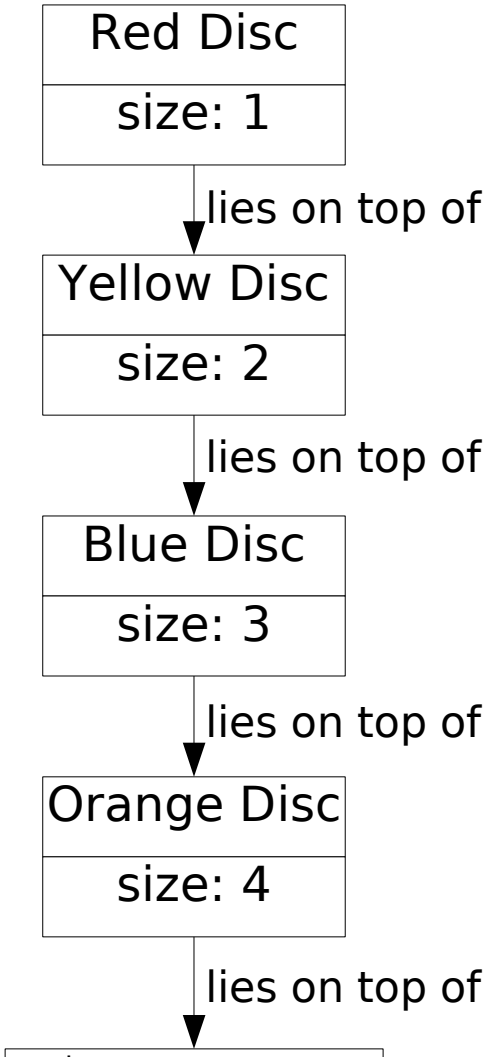
From Objects to Classes

- Group object diagrams under a use cases (make sure to split complex use cases in several)
- Look at all object diagrams of a use case
- Locate groups → classes
- Compare pre- and postconditions → methods
- Read attributes, multiplicity and association kinds directly from object diagram

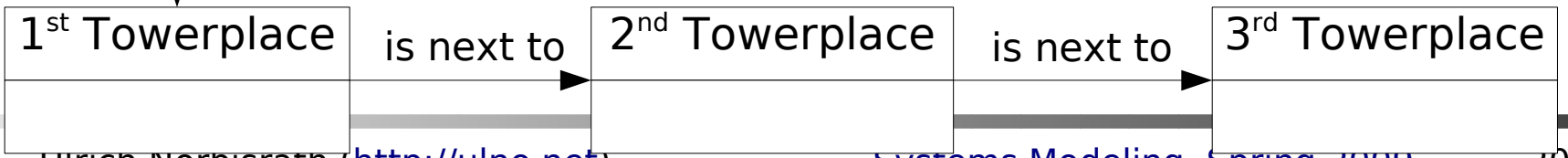
User Story (revisited)

- Title: Move red disk to second place
- Precondition: red d. (size 1), yellow d. (size 2), blue d. (size 3), and orange disc (size 4) sorted on initial towerplace (of 3)
- Action: The player moves the red disc onto the second towerplace
- Postcondition: yellow, blue, and orange disc sorted on initial towerplace (of 3), red on second towerplace

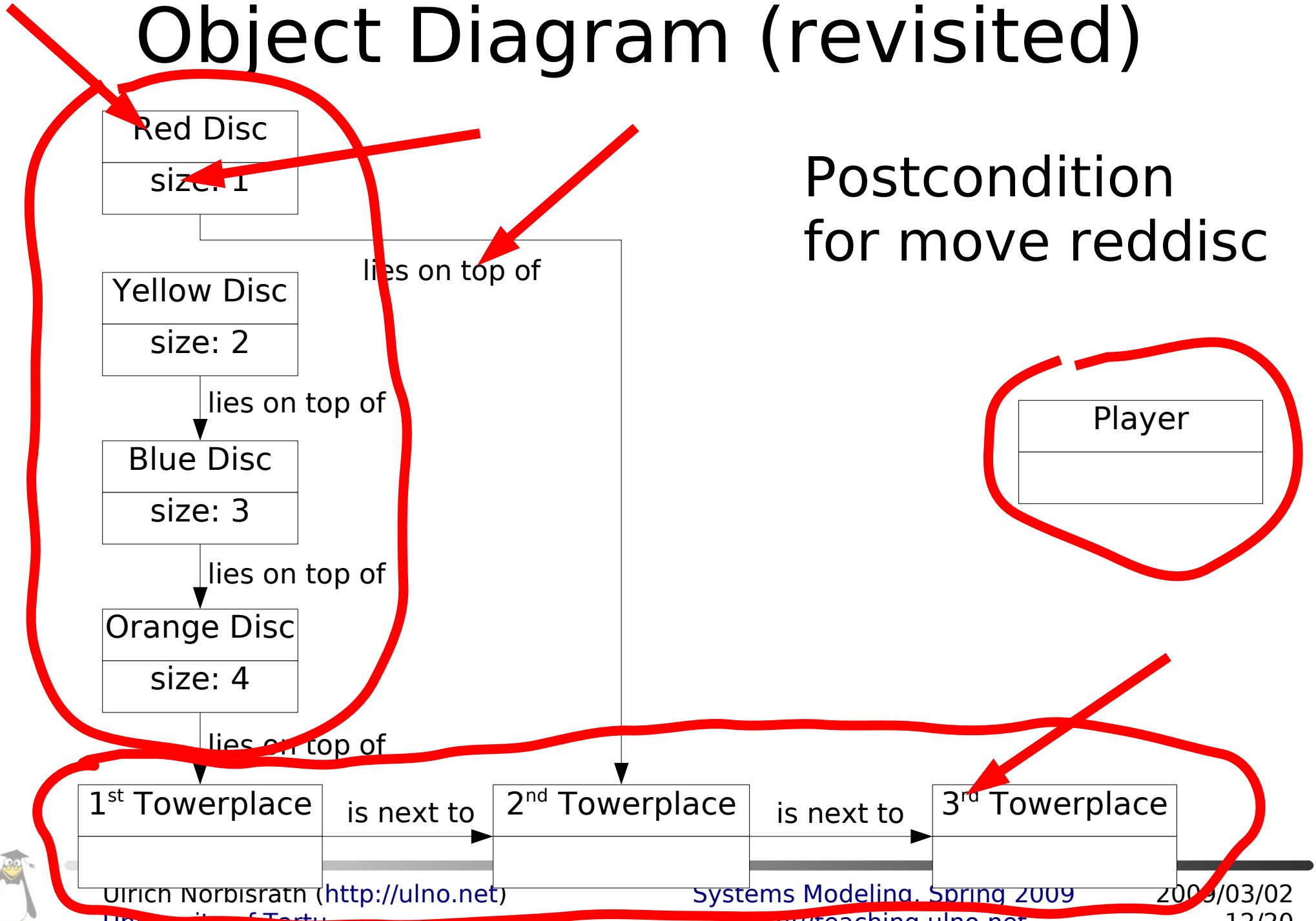
Object Diagram (revisited)



Precondition for move red disc



Object Diagram (revisited)



Postcondition
for move reddisc

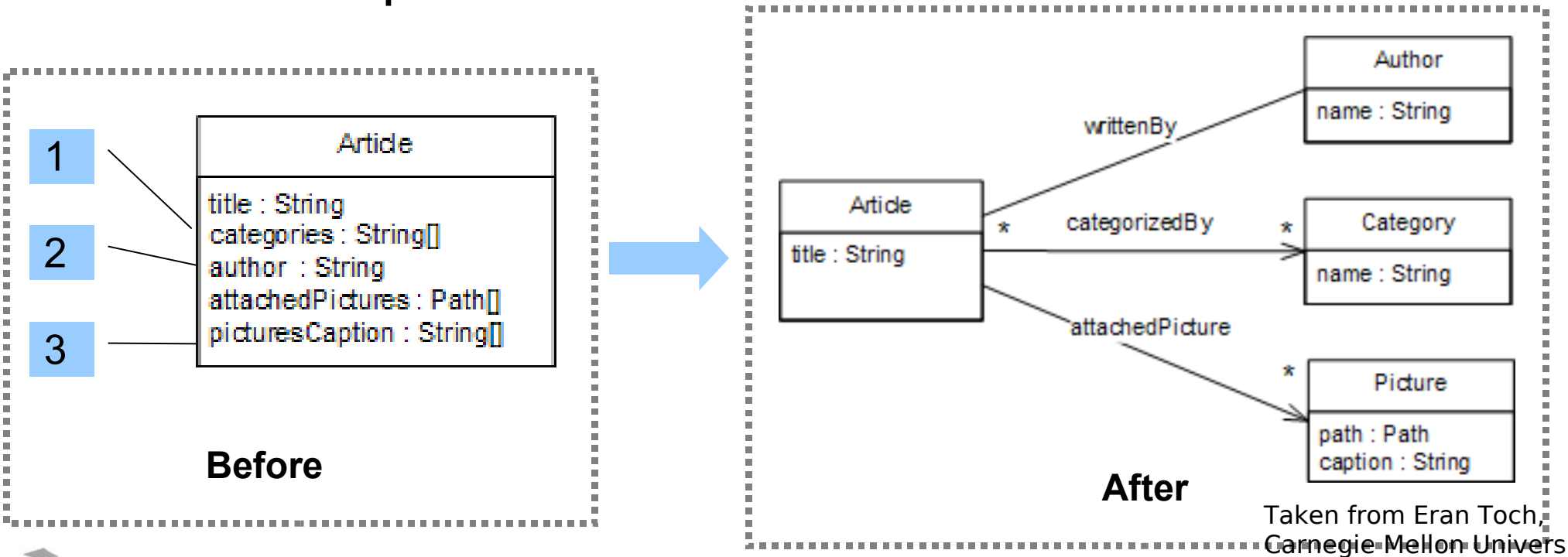
Mini Exercise

- Team up (3-4)
- Draw class diagram for Hanoi in ArgoUML
  (only covering move disc cases)
  (10-15 min)
- Present

Normalization

Classes should be normalized, if:

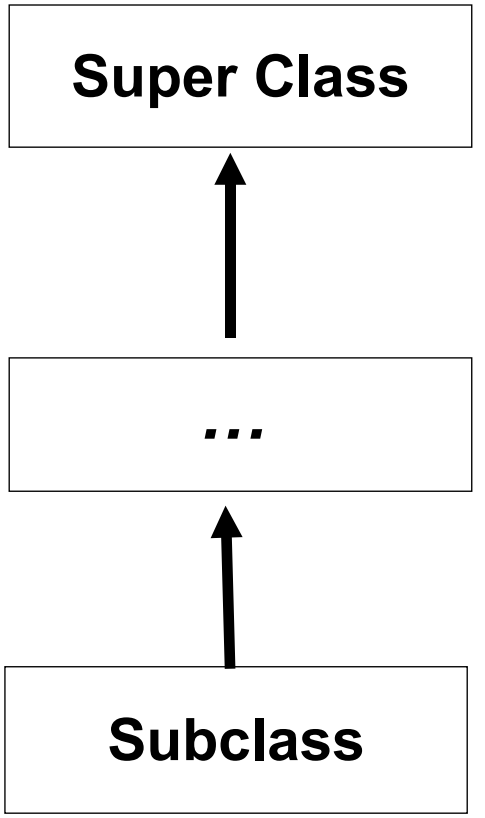
- Attributes are selected from large or infinite sets
- Relations with attributes are in n:n form
- Groups of attributes are related



Taken from Eran Toch, Carnegie Mellon University

Generalization

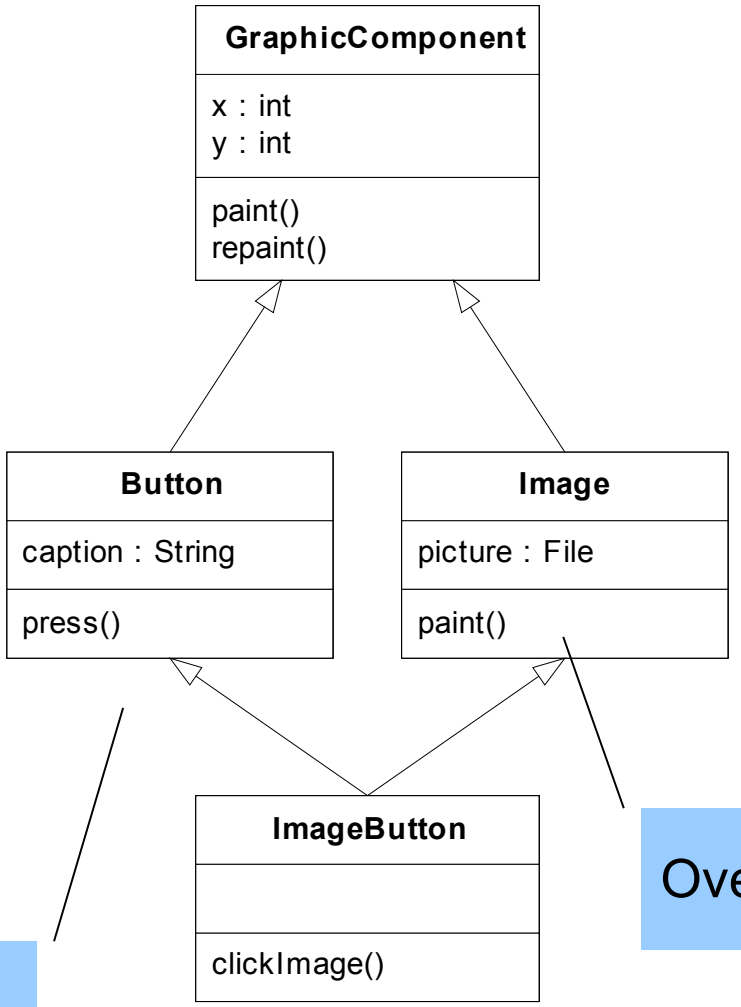
- Super Class (Base class)
  - Provides common functionality and data members
- Subclass (Derived class)
  - Inherits public and protected members from the *super class*
  - Can extend or change behavior of super class by *overriding* methods
- Overriding
  - Subclass may override the behavior of its super class



Taken from Eran Toch, Carnegie Mellon University

Generalization – Advantages

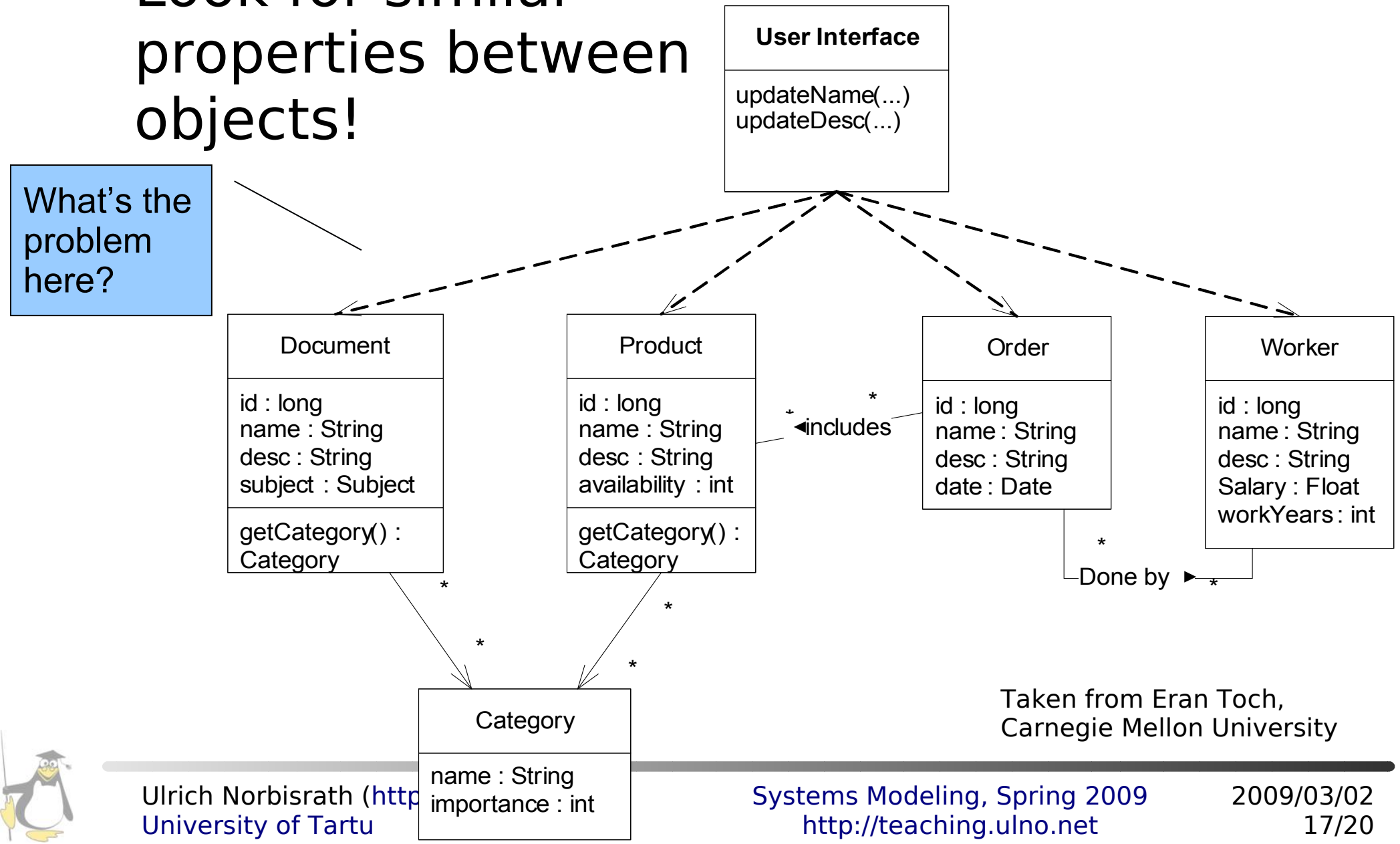
- Modularity:
  - Eliminate the details
  - Find common characteristics among classes
  - Define hierarchies
- Reuse:
  - Allow state and behavior to be specialized



Taken from Eran Toch, Carnegie Mellon University

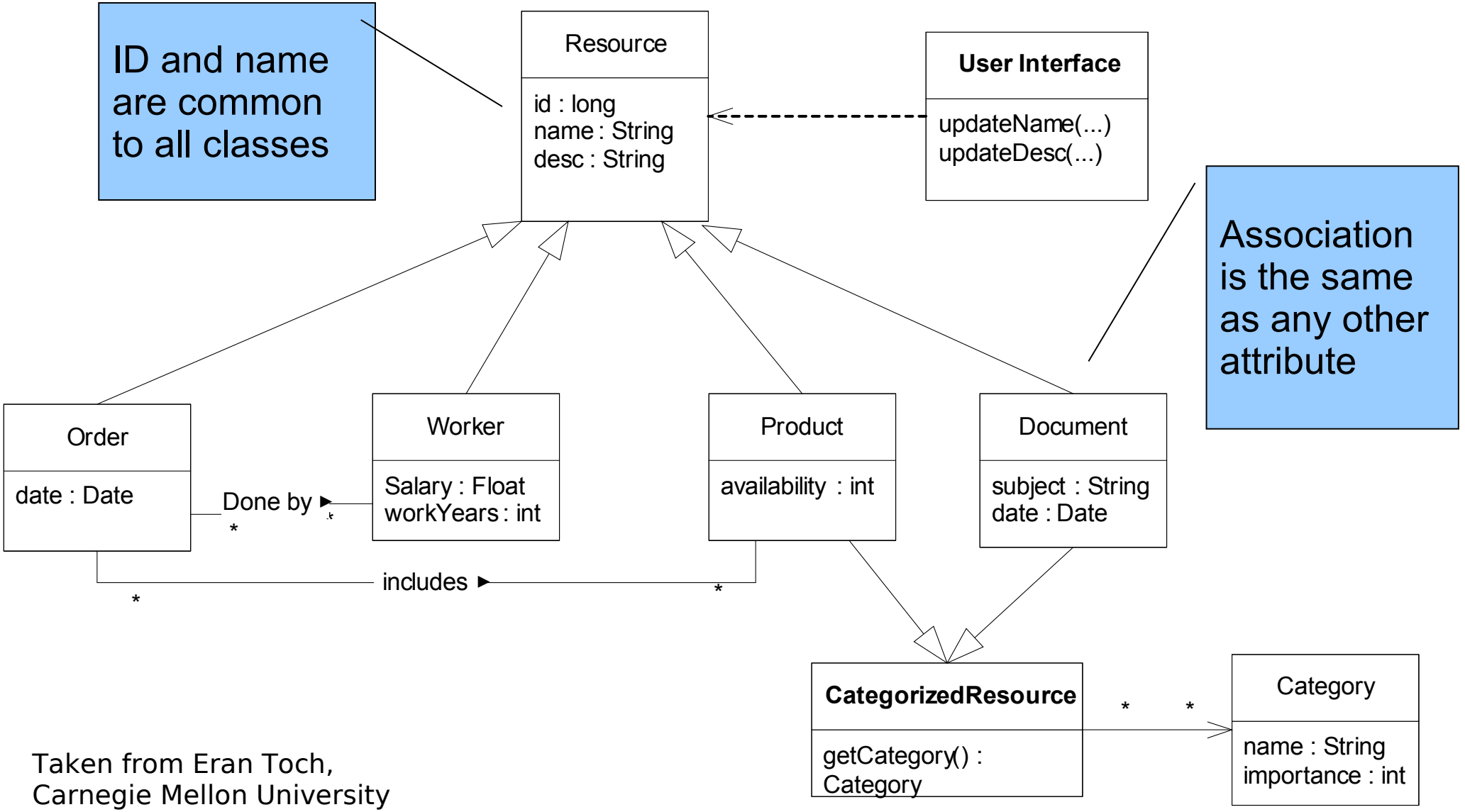
Generalization Guidelines

- Look for similar properties between objects!



Taken from Eran Toch, Carnegie Mellon University

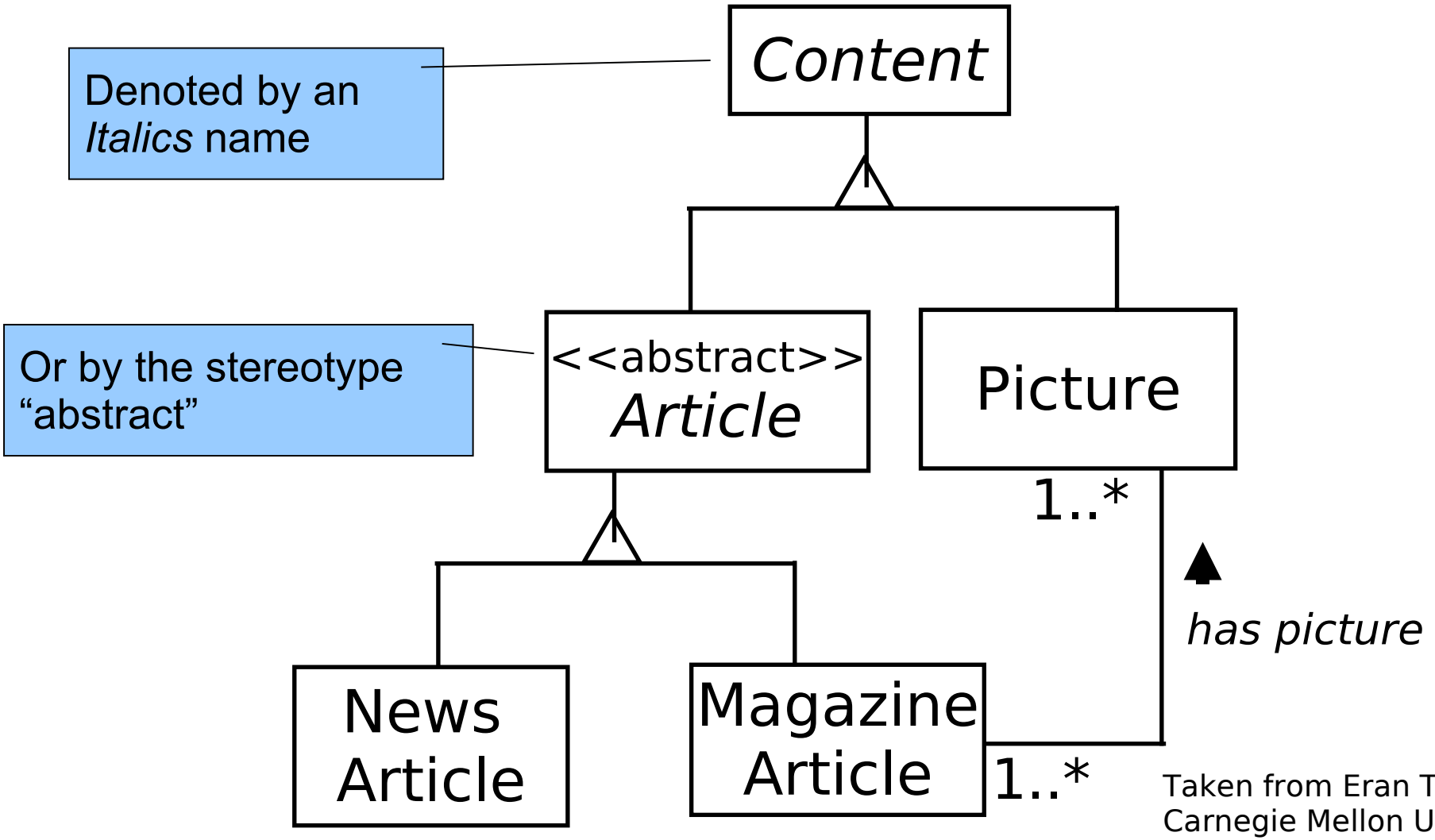
Generalization – continued



Taken from Eran Toch, Carnegie Mellon University

Abstract Class

- A class that has no direct instances



Taken from Eran Toch, Carnegie Mellon University

Lab- and Homework

- Draw class diagram in an UML Modeler for the elsim controller
- Send repository archive or link to me until March 8th