

Software Design
CSE 335, Spring 2006

Static Data Modeling

Stephen Wagner

Michigan State University

Heuristics for designing *good* classes

- Modeling involves *analysis* and *reflection*:
 - Looking for common flaws
 - Probing your design to try to break it
 - Analyzing your model for fidelity and completeness
- How can heuristic methods help?
 - Suggest a regimen of tests to apply to your models
 - Gives you some patterns that serve as “red flags”, to make it easier to spot poor designs

The poorly analyzed class ...

- Suppose we're modeling an enrollment system, and someone suggests:

Course	
name	: string
sect	: int
desc	: string
time	: time
days	: dayname
credits	: int

Technique: Think up instances and pose update questions

<u>:Course</u>	
name =	CSE260
sect =	1
desc =	math
time =	8:00
days =	MW
credits =	3

<u>:Course</u>	
name =	CSE260
sect =	4
desc =	math
time =	4:10
days =	TTh
credits =	3

<u>:Course</u>	
name =	CSE231
sect =	1
desc =	algorithms
time =	8:00
days =	MW
credits =	3

What is required to change the CSE260 description?

Technique: Test under addition

Course	
name	: string
sect	: int
desc	: string
time	: time
days	: dayname
credits	: int

<u>:Course</u>	
name =	CSE335
sect =	1
desc =	software
time =	???
days =	???
credits =	4

- Want to add CSE335:
 - We know desc and credits
 - Have no idea when and where
 - How do you instantiate object?

Problem: Attribute inter-relations

- Attributes related to one another in different ways
 - desc and credits are uniquely determined by name
 - time and day are not uniquely determined by name
 - They are determined by name and section

- We want *cohesive* attributes

Course	
name	: string
sect	: int
desc	: string
time	: time
days	: dayname
credits	: int

Heuristic: Identify candidate key(s)

- Defn: *Candidate key* is a minimal set of attributes that uniquely identifies an object
 - e.g., social-security number for Person, airportCode for Airport, etc.
 - Attributes also uniquely determined by candidate key
- Heuristic: If attributes a_1 and a_2 are determined by different keys k_1 and k_2 , then a_1 and a_2 should probably be in different classes with keys k_1 and k_2

Solution: Two related classes



We discovered a new class

“Red flags” that signify poor analysis

- Data model with *few* classes and associations
 - Real world data is rich and varied
 - So we expect a model of such data to exhibit this richness and variation
- Classes with lots of attributes
 - Especially when most attributes are of type string
 - Problem: Strings may represent “object identifiers”
 - * But they do so poorly
 - * Associations better at relating objects

Example: Lots of string attributes

Professor	
name	: string
office	: string
department	: string
chair	: string
fax	: string

Which of these attributes are legitimate strings, and which are object identifiers?

Exercise: Redesign Professor class

- How should we design the professor class?

Look for the *unseen* classes

- Discovery often leads to elegant designs: e.g., class *offering*
- Attribute/candidate-key analysis is a “bottom-up” method of discovery
- Top-down technique: Look for *higher-degree associations*:
 - Associations often have degree greater than 2
 - Often, there is information associated with “a relation” as opposed to with a particular class

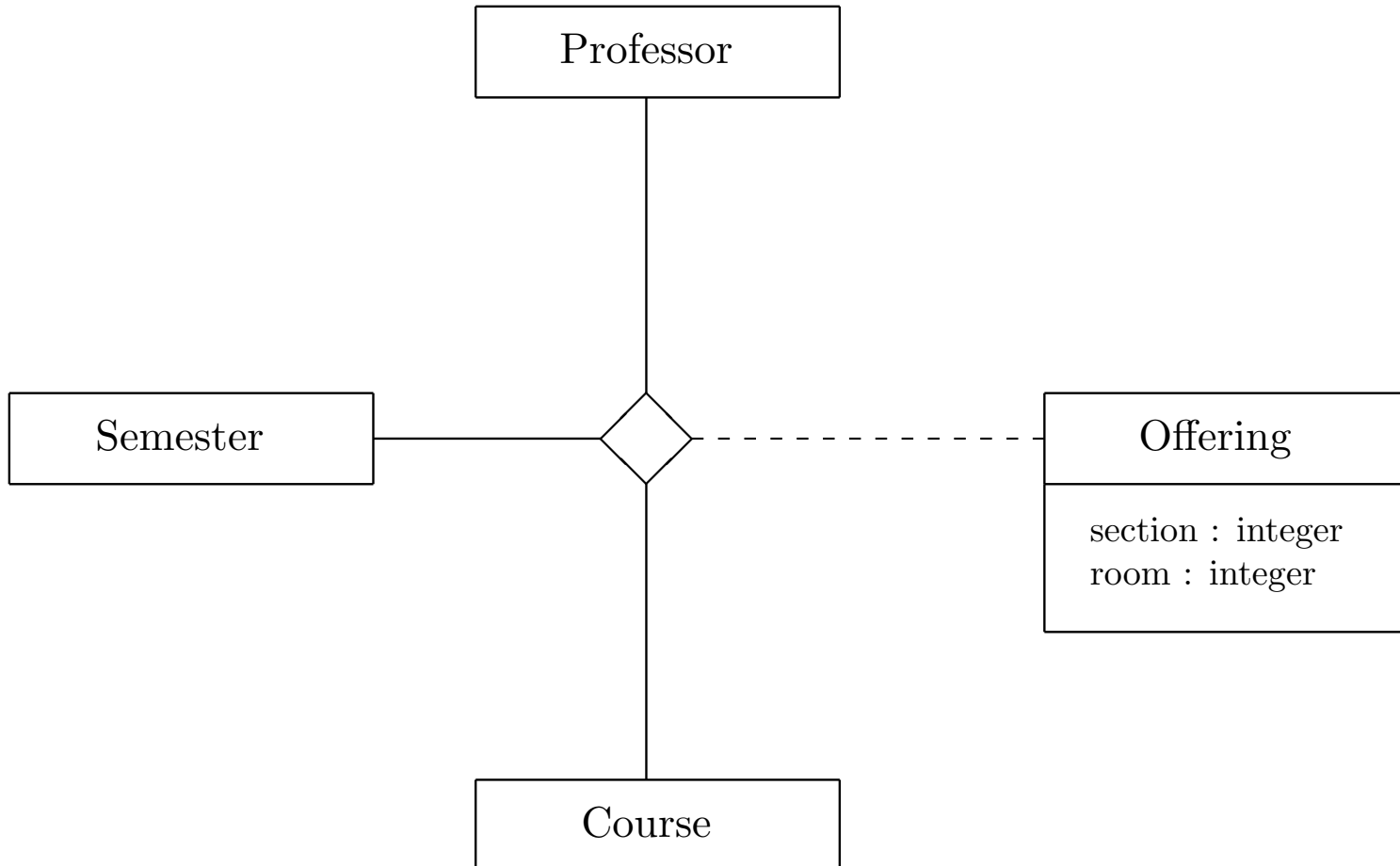
Example: Enrollment System

Professor

Semester

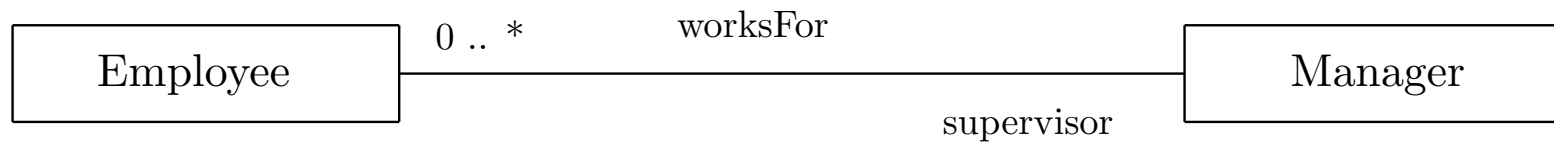
Course

Solution



Classes vs. roles in associations

- Defn: A *role* is one end of an association
 - May be given an explicit name
 - Always a noun

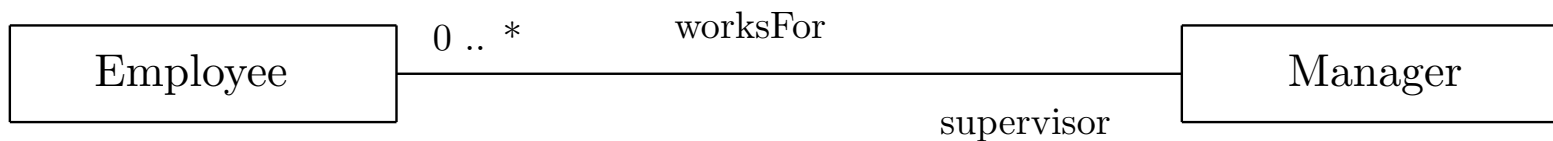


- Often easy to confuse a class with a role
- Analysis aims to remove this confusion

Role/class analysis

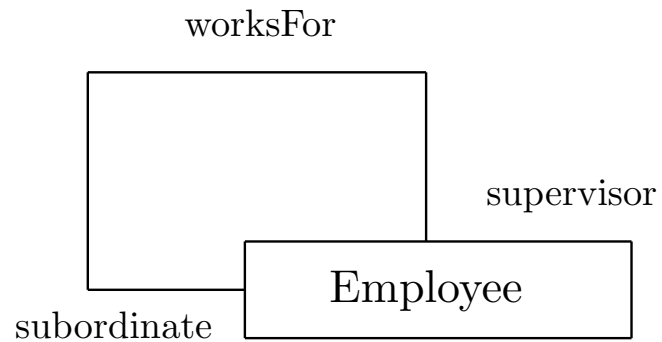
Questions

- Isn't a Manager an Employee?
- Isn't there a "top-level" Manager who has no supervisor?
- Can employees have multiple supervisors?
- Can an Employee ever be promoted to a Manager?



Reflexive associations

Association both of whose roles are played by objects of the same class

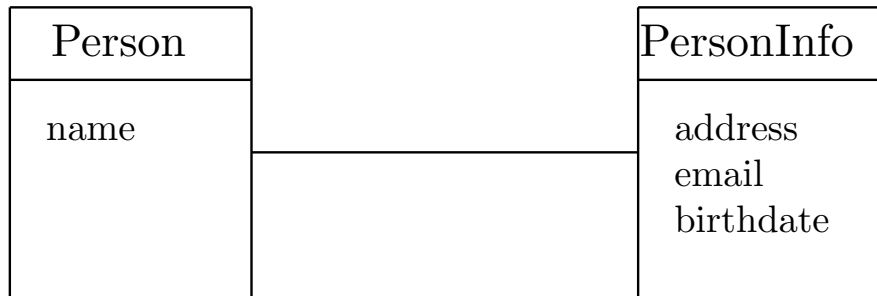


What are the multiplicities?

Can an employee be promoted to a supervisor?

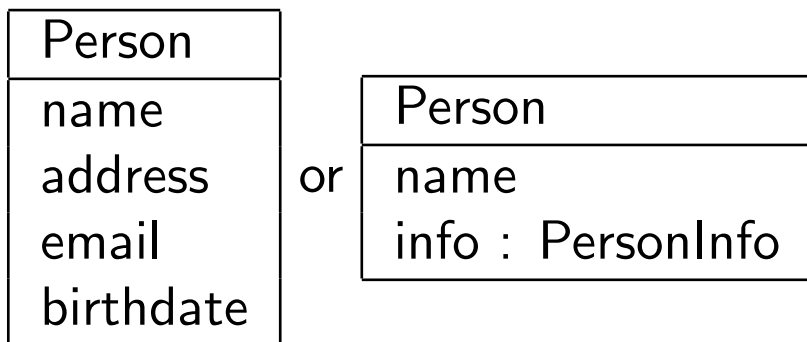
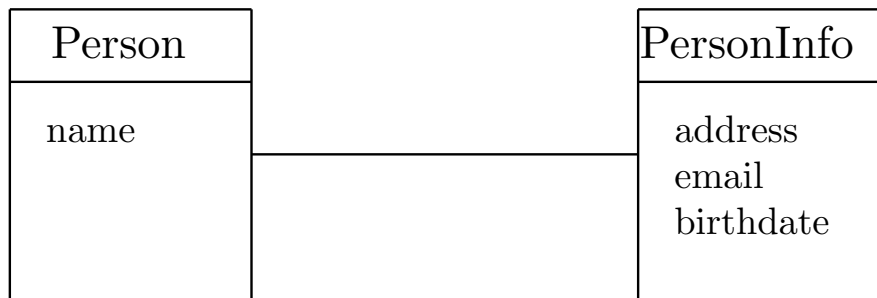
Implications of 1 to 1

1 to 1 \Rightarrow create (delete) one, have to create (delete) other

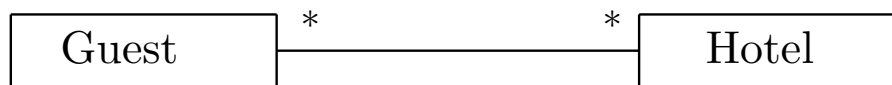
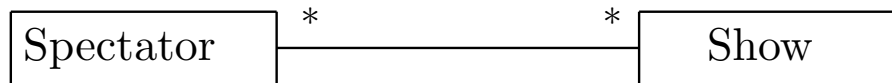
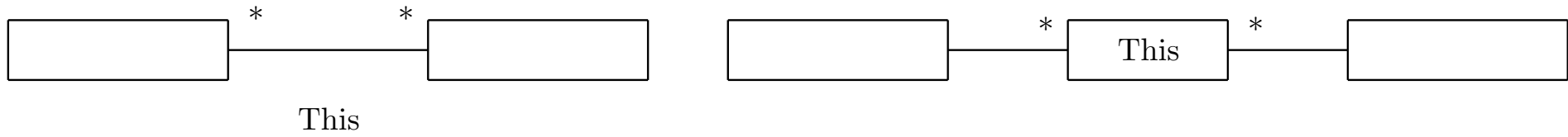


Implications of 1 to 1

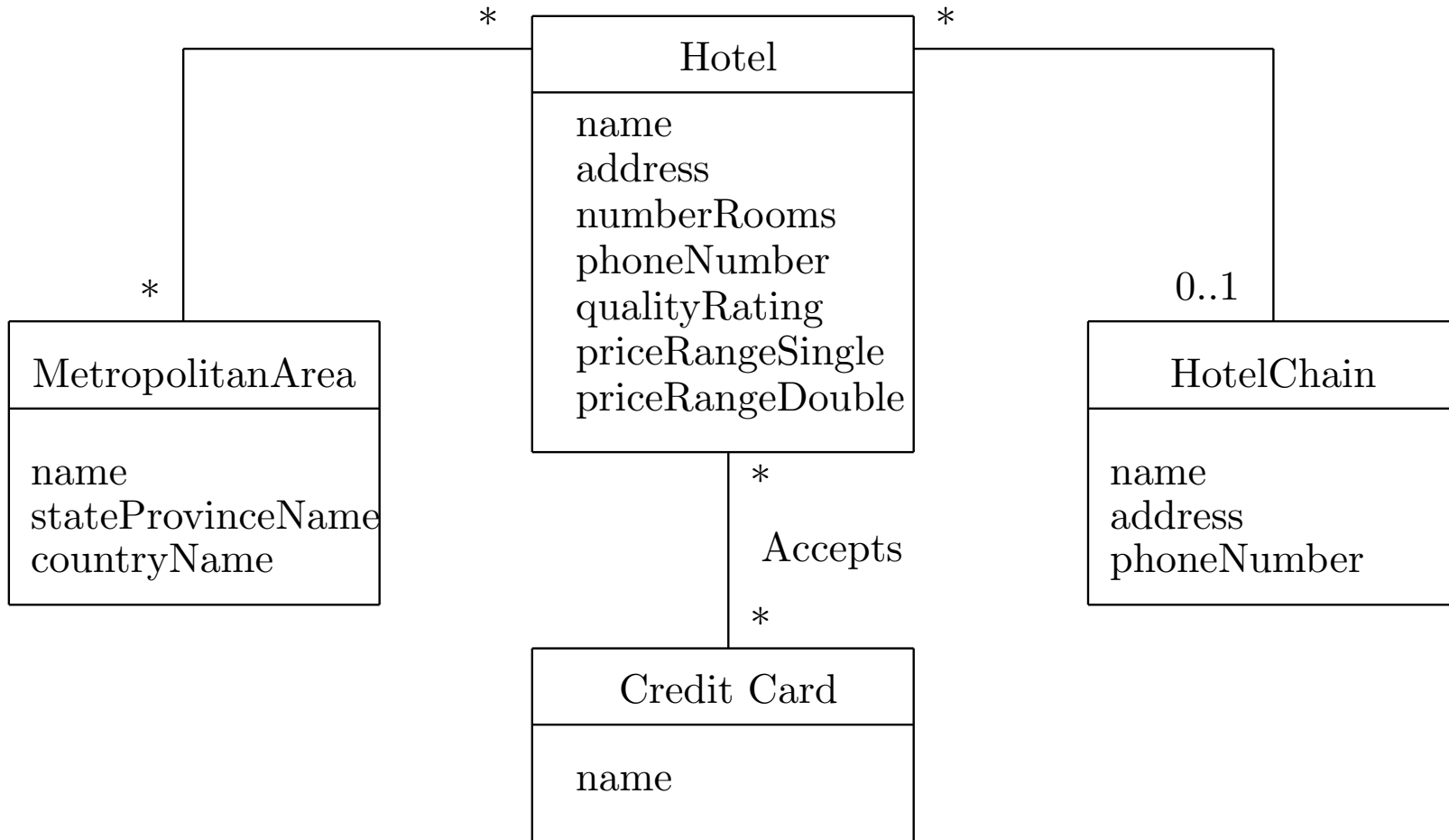
1 to 1 \Rightarrow create (delete) one, have to create (delete) other



Many to Many Transforms



Example: Hotel Selection



Observations

- Credit Card modelled as a class, rather than an attribute. Why?
- Distinction in modeling process
 - usually an iterative process
 - instance diagrams can help discover limitations or unnecessary constraints in the model
 - can work from the bottom up, or the top down
- No references to initialization (e.g. no constructors)
- Pure data; no references to presentation or collection classes (e.g. no buttons, menus, etc.)

Exercise

Create a static data model for an airline reservation system with the following types of data:

- Airline
- Airport
- City
- Flight
- Pilot