Developing High Class UML Class Models

Jeff Jacobs

Covad Communications

jmjacobs@jeffreyjacobs.com

Survey

- Who is familiar with UML class models?

- Who creates class models?

- Who reads/reviews class models?

- Developers?

- Modelers?

- Analysts?

- Familiar with "logical" E/R modeling techniques?

Agenda

- Why model?
- Rantings of a lunatic presenter
- Quick review of UML class constructs
- Rules, guidelines, recommendations
- Summary

Disclaimer

- The views presented here are those of the presenter and do not represent those of:
  - Oracle
  - ODTUG
  - Authors of any UML books
  - Any standards group
  - Any other internationally recognized self proclaimed UML guru

Presenter Biases

- Presenter is biased toward
  - Completeness
  - Understandability
  - Correctness
  - Communication with non-technical parties
  - Using appropriate tools and techniques
  - Disciplined thinking
  - Modeling as a process
- *E/R quality tests and metrics are directly applicable to class models!*

Why do we model?

- Understand the world/domain/issue
- Create a representation of reality
- "Requirements"
- Generate code
- Design database
- Design code

What are we addressing?

- Classes for "analysis" (not code design)
  - Requirements
  - "Data"
  - "Real world"
  - Architecture
  - "Domain"
- Classes
- Associations
- *Not attributes, methods or responsibilities*
- *Not code design or reverse engineering*

The Rants of a Lunatic Presenter

- Most UML class diagrams are:
    - Sloppy
    - Imprecise
    - Incorrect
    - Incomplete
    - Misleading
- Do not accurately reflect
    - The "real" world
    - Needs of the business
- Barely understandable

Why?

- Most class models are created by "system" architects" or developers
  - Many architects aren't
- Constant confusion of "modeling" with implementation
- Class models used for anything and everything
- No fundamental theory or good practices
  - (unlike E/R modeling)
- Too many constructs in toolbox
  - Notation by acquisition
  - "If it's there, I should use it"

Class Basics

- **Classes (boxes)**
  - Similar to entities
- **Classes have attributes and operations (methods)**
- **Associations (lines)**
  - "relationships"
  - Navigation
    - Association can only be traversed in one direction
  - Dependency
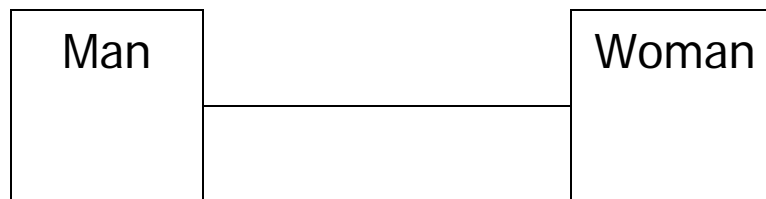 - Generalization (inheritance)
  - Aggregation
  - Composition

Class Basics

- ## Association may have "adornments"
    - Name
    - Role
    - Multiplicity
    - Aggregation
    - Composition
- ## Association classes
    - Hybrid between classes and associations

The Problem

- What does this mean?

```
┌──────────┐              ┌──────────┐
│ Man      │              │ Woman    │
│          │──────────────│          │
│          │              │          │
└──────────┘              └──────────┘
```
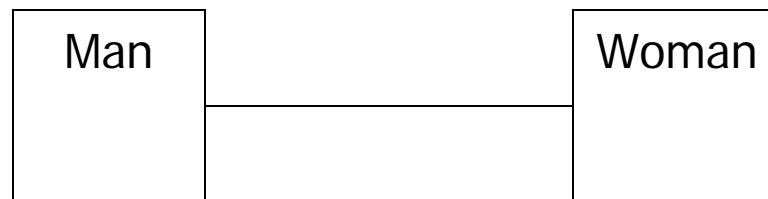
What is Quality?

- Understandable to all interested parties

- Unambiguous

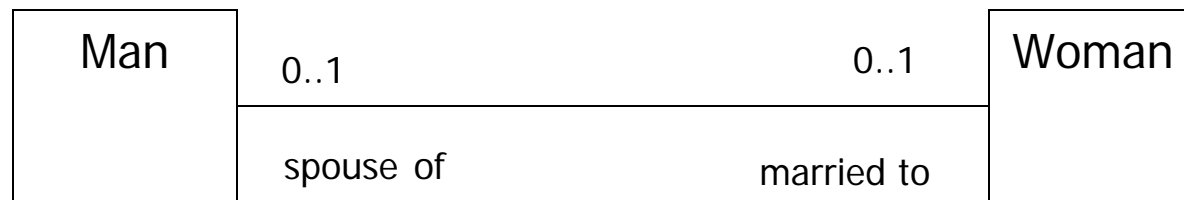- "Complete"

- Correct

- Appropriate level of abstraction

A Line is Just a Line

```
┌─────────┐              ┌─────────┐
│  Man    │──────────────│  Woman  │
│         │              │         │
└─────────┘              └─────────┘
```

But a Relationship is a Thing of Beauty

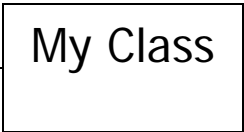| Man	0..1		0..1	Woman
	spouse of		married to	

Rule 1 – Explicit Multiplicity

- Pop Quiz!

- Is ———————————— 1 | My Class |

    - 0..1
    - 1..1
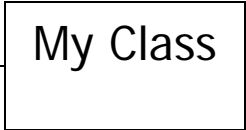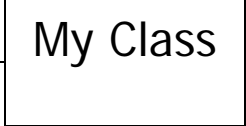
Rule 1 – Explicit Multiplicity

- Pop Quiz**!**
- Is ————————1 | My Class |
  - 0..1
  - 1..1
- Is ————————* | My Class |
  - 0..*
  - 1..*
- Explicit is better
  - Everybody is clear

Rule 2 – Name that Association

- Associations may have one "association name or a role name for each end of the association (or both)"

Man		Woman

0..1　　　　　marriage　　　　　0..1

spouse of

married to
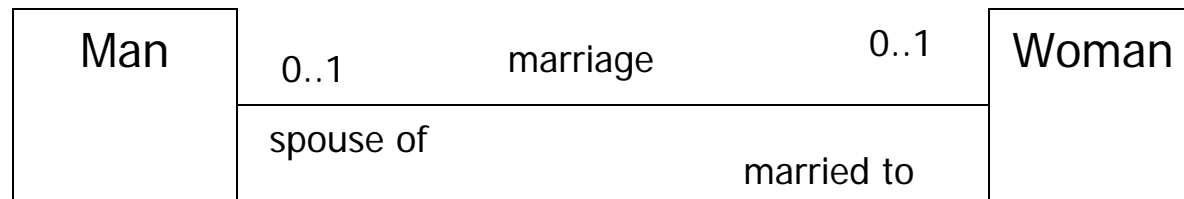
Rule 2.1a – Use ▶ for Association Names

- Who sends/receives?

| Server | 1..1   receives from   0..* | Client |

Rule 2.1a – Use ▶ for Association Names

```
┌─────────┐                                      ┌─────────┐
│ Server  │  1..1    receives from ▶    0..*     │ Client  │
│         │──────────────────────────────────────│         │
│         │                                      │         │
└─────────┘                                      └─────────┘
```
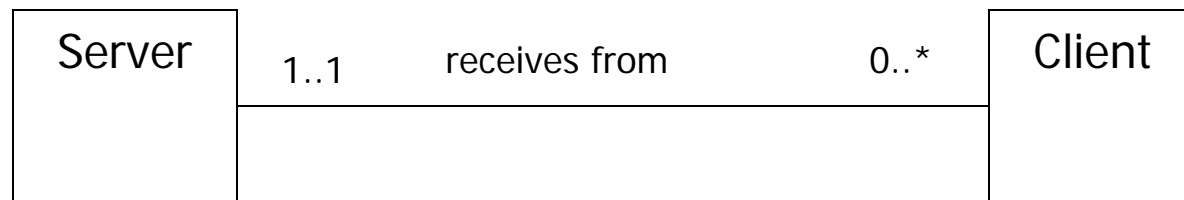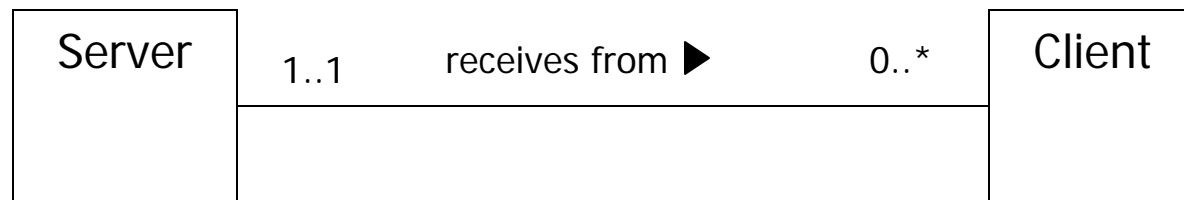
Roles Preferred

- ## Roles preferred
  - ### Reduces ambiguity
  - ### Easier to read in both directions

Server		Client
1..1		0..*
receives from		sends to

Rule 3 – Use "Good" Names

ProductReferences

0..n   0..n

Product

0..1

0..n   0..1

ProductBundleComprisedOf

0..1   ProductBundle   ProductComponent

ProductHasCustomerFacingServices

ProductHasPhysicalResources

Service

Resource

0..n   0..n

ResourceFacingService   *CustomerFacingService*   *LogicalResource*   *PhysicalResource*

0..1 0..1 1..n   0..n   1..n 0..n   0..n 1..n

CFServiceRequiresRFServices

PResourceSupportsLResource

LogicalResourcesImplementRFS

PhysicalResourcesHostRFS

Rule 3.1 – Understandable and Readable

- *married to/spouse of*

Rule 3.1 – Understandable and Readable

- *married to/spouse of*
- *provided by* ▶

Rule 3.1 – Understandable and Readable

- *married to/spouse of*

- *provided by* ▶

- Avoid
  ReallyLongNamesWithCapitalizationBecauseTheyAreHardToReadByMortals

- Use spaces or underscores (CTW), because
  Really long names with capitalization they are hard to read by mortals

Rule 3.1 – Understandable and Readable

- *married to/spouse of*

- *provided by* ▶

- Avoid
  ReallyLongNamesWithCapitalizationBecauseTheyAreHardToReadByMortals

- Use spaces or underscores (CTW), because
  Really long names with capitalization are hard to read by mortals

- Avoid redundancy and confusing/meaningless names:

Product Offering	ProdOfferingReferencesProdSpec	Product Specification

Rule 3.2 – Include Definitions/Descriptions/Comments of Classes and Attributes

- ## The diagram is not the model
  - ### It is only a representation
- ## Class names are seldom sufficiently descriptive

Rule 4 – Use E/R Reading Conventions

- 0.. = "May Be"

- 1.. = "Must Be"

- Helps ensure correctness of optionality
  - Enforces "discipline" and consistency
  - More acceptable to non-techies than "zero or more"

- Whichever reading technique you choose...
  - *Be Consistent!!!*

Rule 5 – Resolve Many to Many Relationships
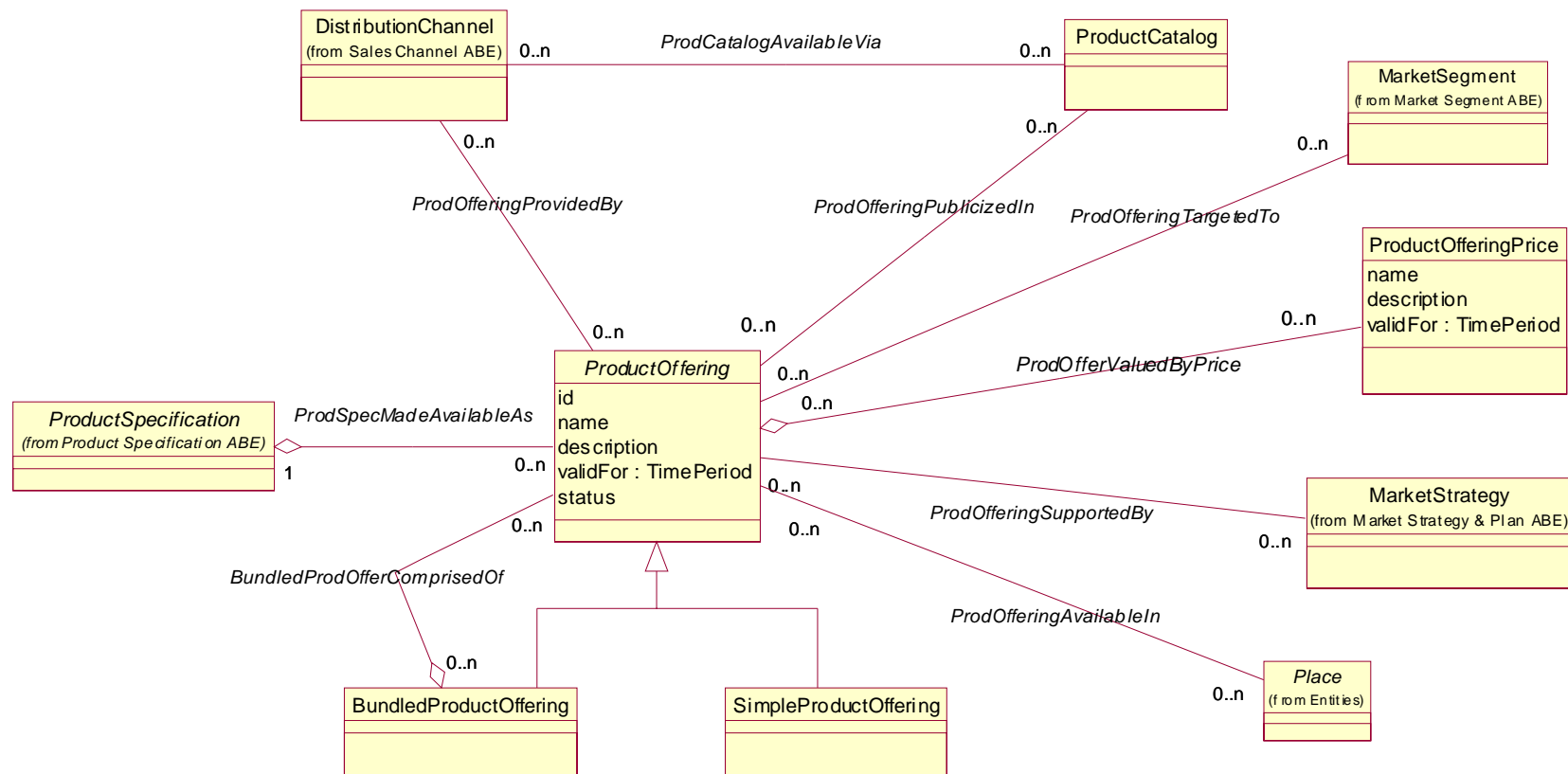
Rule 5 – Resolve Many to Many Associations

- M:M relationships "hide" important detail that must be discovered

- M:M produce brittle implementations

- M:M result in weak Object/Relational mappings

- M:M relationships should be eliminated by end of detailed "domain" analysis

- Iterative process of refinement

| Employee	Assigned to	Worked by	Project
	0..*	0..*	
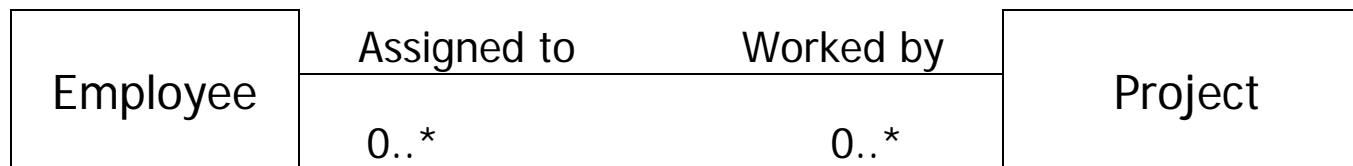
Resolving Many to Many Associations

- To resolve a M:M association:
    - 1) Create new class (*not an Association Class)*
    - 2) Create associations back to original entities
    - 4) Use meaningful names for new entity and relationships
    - 5) Examine new entity for attributes and relationships

Resolving Many to Many Associations

- Create new class
- *New name is very important!*
- What would be a good name?

Employee	Assigned to Worked by	Project
	0..* 0..*	
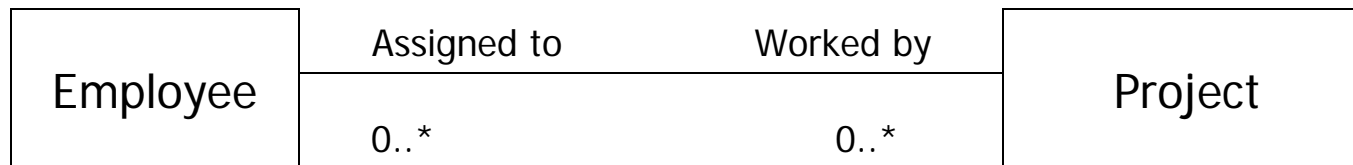
Name New Class

- *Usually found in original role/association name!*
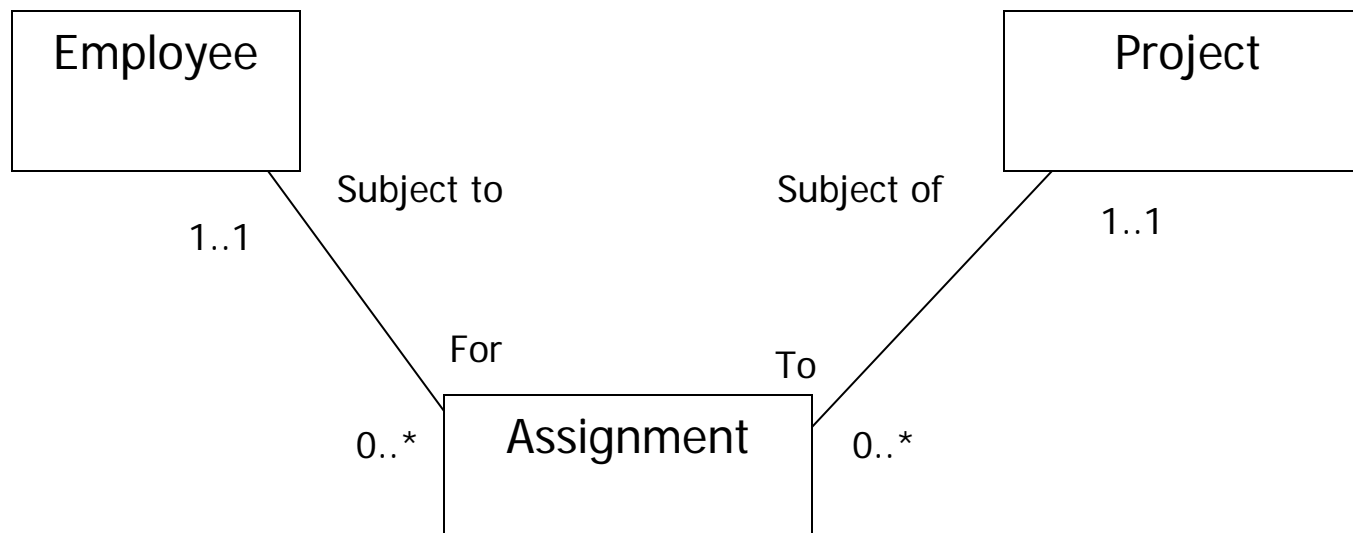
Employee		Project
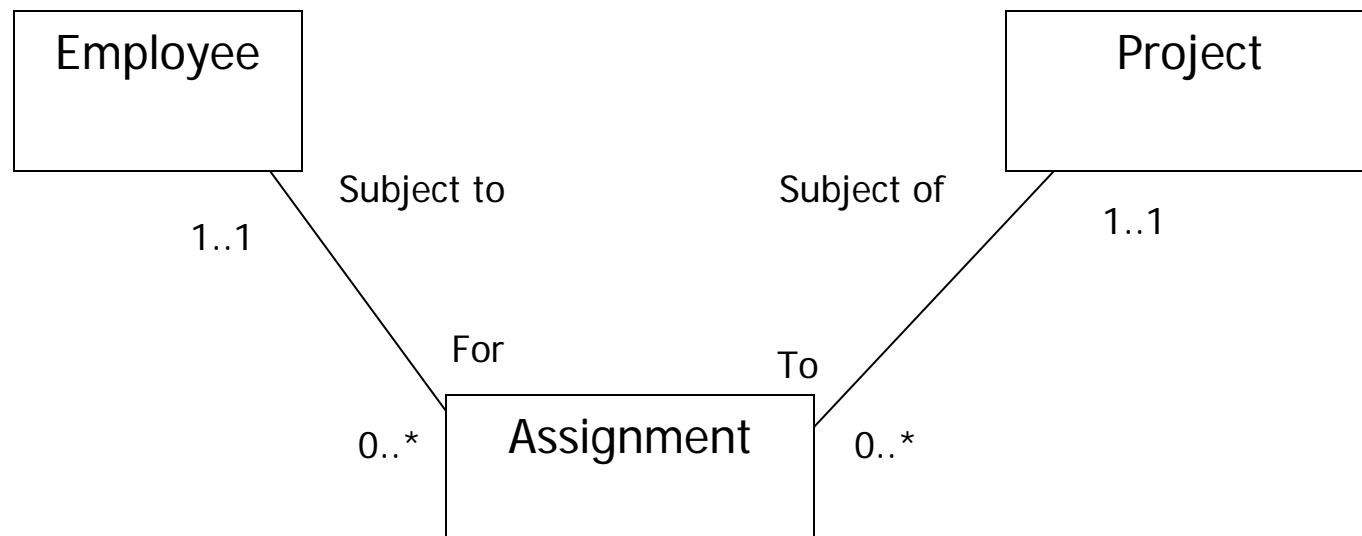
Subject to   Subject of

1..1   1..1

For   To

0..*   **Assignment**   0..*

Examine New Class for Attributes and Associations

Employee		Project

Subject to   Subject of

1..1   1..1

For   To

0..*   Assignment   0..*

Examine New Class for Attributes and Associations

```
┌─────────────────┐                              ┌─────────────────┐
│   Employee      │                              │     Project     │
│                 │                              │                 │
│                 │                              │                 │
└─────────────────┘                              └─────────────────┘
         \     Subject to          Subject of      /
      1..1 \                                       / 1..1
            \                                     /
             \  For              To              /
              \                                 /
          0..* \ ┌───────────────────────┐ 0..*
                 │ Weekly Assignment     │
                 ├───────────────────────┤
                 │ Week Ending           │
                 │ Hours Billed          │
                 └───────────────────────┘
```
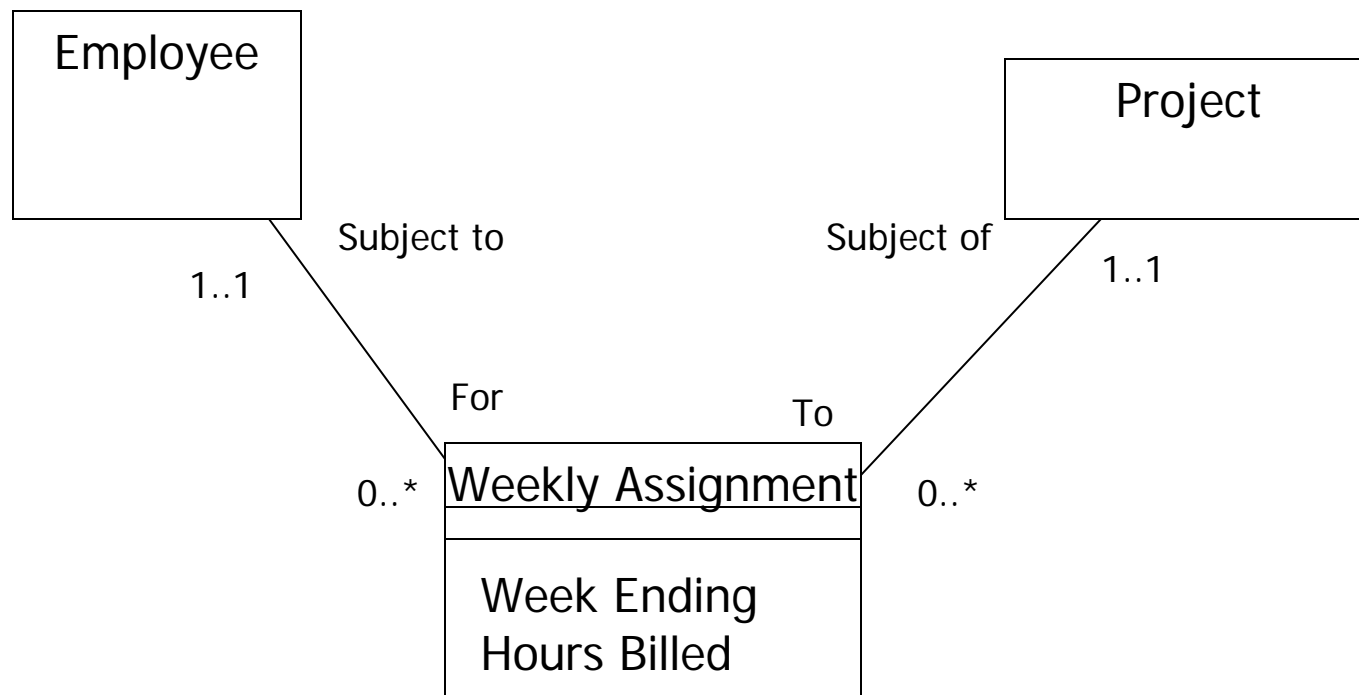
Rule 5.1 – Eschew Association Classes

- Association classes = "association properties"
  - "It wouldn't be appropriate to model this situation with a Company to Job Association together with a Job to Person association"

- In fact, it *would be appropriate!!!*

| Company | * | | | 1..* | Person |

employer   employee

Job
description date hired salary
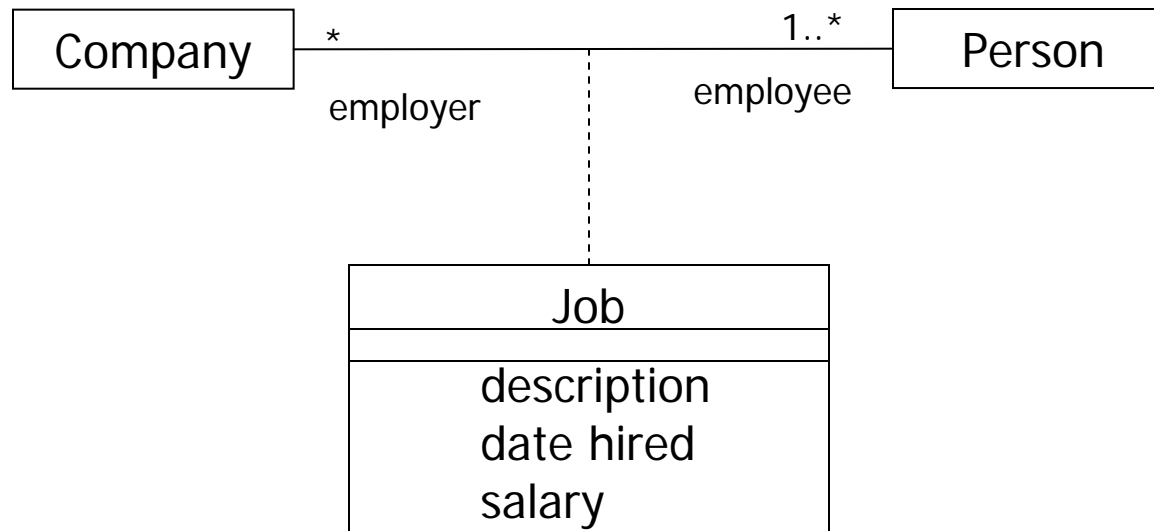
Rule 5.1 – Eschew Association Classes

- **Confusing to end users**
- **No real programming language support**
- **No significant (real?) difference from real classes**
- **Can't be reused**
  - Can't be attached to more than one association

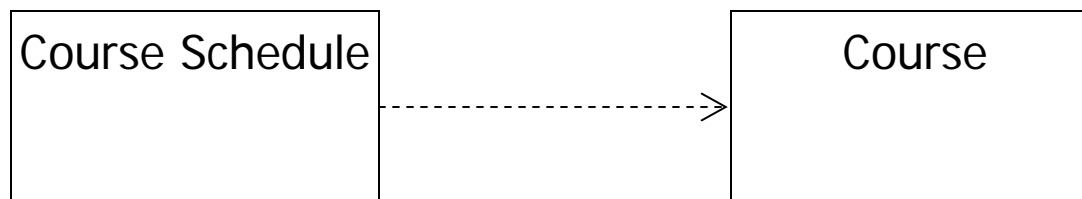# Rule 5.1 – Eschew Association Classes

- **Model as regular classes**
  - Less confusing to business
  - Leads to better analysis
    - No need to "convert" if/when a new meaningful association is discovered
  - Leads to better code
- **(Fill in names and multiplicity)**

| Company	Job	Person
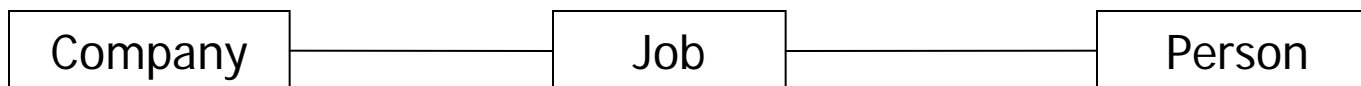
Rule 6 – Avoid Dependencies

- *"A semantic relationship between two things in which a change to one (independent thing) may affect the semantics of the other (dependent thing)"*
- Generally meaningless except in code design
    - "Input parameters"
- *"If you provide the full signature, you don't normally need to show the dependency"*

```
+----------------------+          +----------------------+
|   Course Schedule    | - - - -> |       Course         |
|                      |          |                      |
|                      |          |                      |
+----------------------+          +----------------------+
```

Rule 6.1 – Avoid Navigation

- Seldom meaningful; usually clear from context
- Association roles/naming better and clearer
- Constrains implementation
  - (if anybody pays attention)
- Frequently incorrect

SetTopController	→	PowerManager
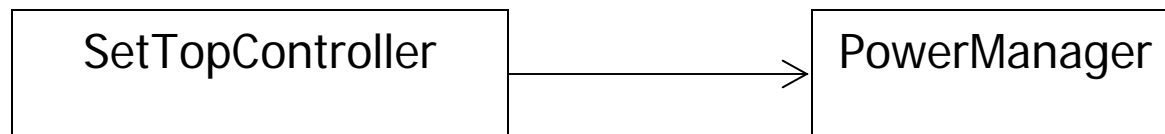
Rule 7 – Use Aggregation Sparingly

- "Simple aggregation is entirely conceptual and does nothing more than distinguish a 'whole' from a 'part'"
  – *The UML User Guide*

- No real semantics

- Easily misused and confused with "composition"
  - Even the UML User guides mixes them up in the same chapter!!!

- Clearly stated relationships are usually better
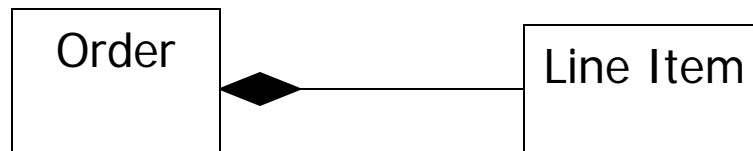
Example (*The UML User Guide*)

- What does "aggregation" add?
- (Tool doesn't support ▶)
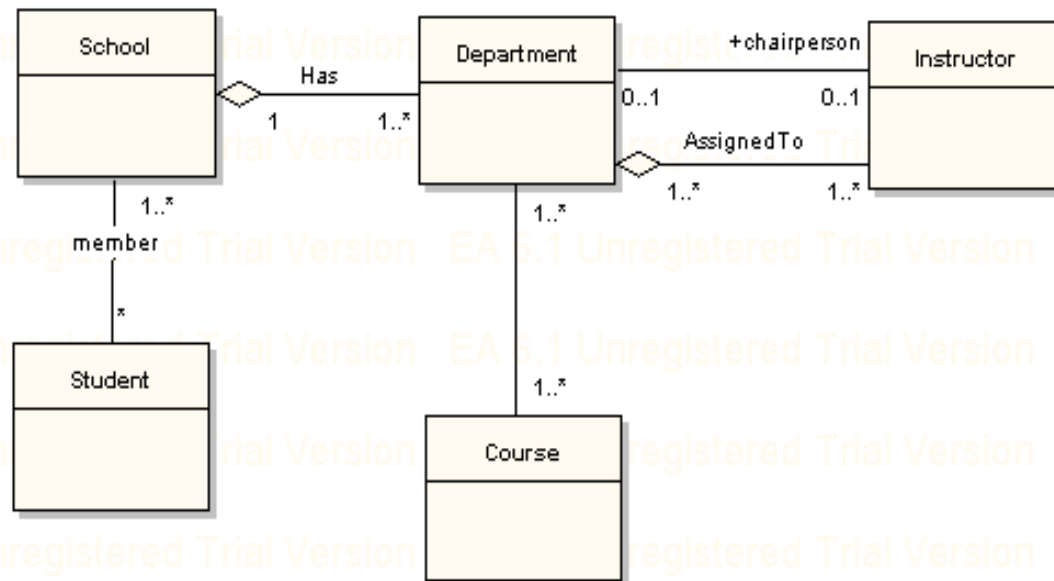
Rule 8.1 – Be Careful with Composition

- Composition has well defined semantics
- Existence of child depends on existence of parent
  - "Cascade delete"
- Only one parent allowed
- Use only when appropriate

Order	◆———	Line Item

Rule 8 - Avoid N-ary Associations

- *"An association among 3 or more classes"*
- Abandoned by the ER community years ago in favor of binary associations
  - Very confusing
  - Seldom informative
- Represent as a class
  - There will always be attributes
  - There will always be more things to discover
- Implementation will be a "class" (or table)

Rule 9 – Be Stingy with Objects

- Objects are *instances of classes*
- Seldom appropriate for "analysis"

Elyse

Rule 9 – Be Stingy with Objects

- **Use sparingly**
- **Specify class**

Elyse:Customer

Rule 10 - Get the "Optionality" Correct

- "1.." vs "0.."

- Most common "mistake"
  - Found in many standard books

- If "1..", then *must* always be present
  - Are there any exceptions?
  - Don't assume it doesn't matter; *some* programmers will enforce the rule
  - Most don't...

- Can severely impact data base design if incorrect
  - Data modelers/database designers *will* enforce optionality

Rule 10 - Get the "Optionality" Correct

- New department?
- New School?
- Can't be chair of more than one department?
  - Interim
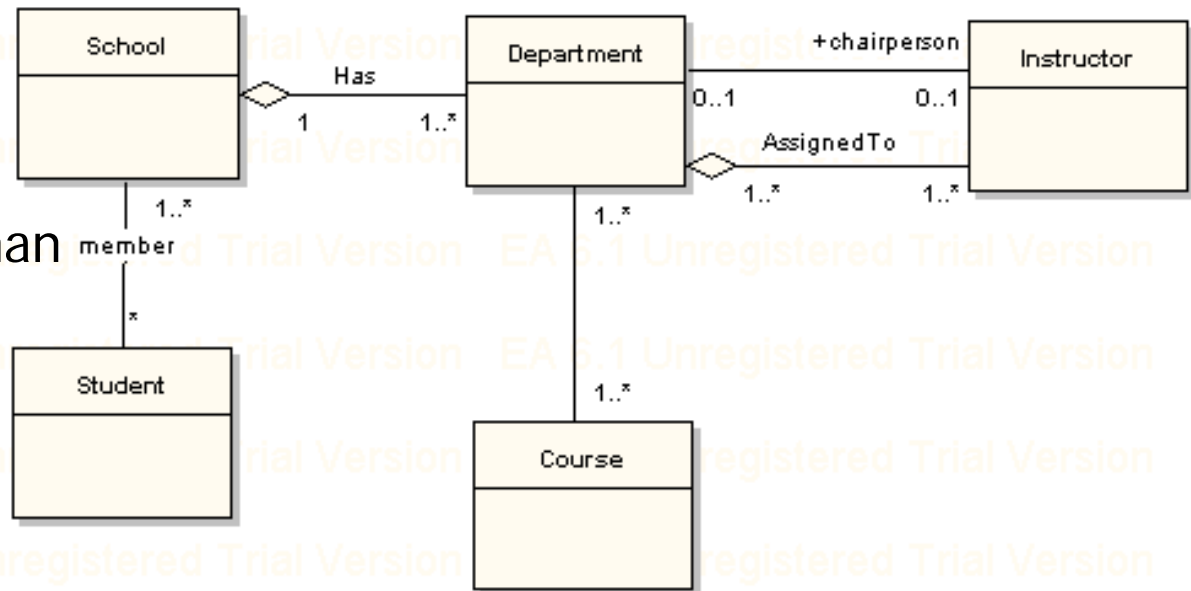
Rule Summary

- **Explicit Multiplicity**
- **Name Associations, preferably with roles**
- **Use meaningful, descriptive names for classes, associations and attributes**
- **Resolve many to many associations**
- **Eschew**
    - Association Classes
    - Dependencies
    - Navigation
    - N-ary associations
- **Use Aggregation sparingly**

Rule Summary

- **Be careful with Composition**
- **Be sure that the semantics are what is truly intended**
  - Don't confuse with aggregation
- **Get the optionality correct**
  - Look for exceptions
  - Be sure the business case is correct and meaningful

Summary

- Class models can be
  - Understandable
  - "Rigorous"
  - Complete
  - "Correct"
- Simple "rules" substantially improve class models
  - Less confusion and ambiguity
  - Better communication
  - More effective
  - Enforce "disciplined" thinking
- E/R discipline and quality techniques can and *should* be applied to class models

Questions and Promo

OPP 2007

February 28 – March 1, 2007

San Mateo Marriott

San Mateo, California

An ODTUG SP* Oracle PL/SQL Programming Conference

*SP – Seriously Practical Conference

ODTUG Kaleidoscope

June 18 – 21, 2007

Pre-conference Hands-on Training - June 16 – 17

Hilton Daytona Beach Oceanfront Resort

Daytona, Florida

WOW-Wide Open World, Wide Open Web!

For more information visit www.odtug.com or call 910-452-7444