

CSSE 374: Design Class Diagrams



Shawn Bohner

Office: Moench Room F212

Phone: (812) 877-8685

Email: bohner@rose-hulman.edu



ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

MY HOBBY:

EMBEDDING NP-COMPLETE PROBLEMS IN RESTAURANT ORDERS

CHOTCHKIES RESTAURANT

~ APPETIZERS ~

MIXED FRUIT	2.15
FRENCH FRIES	2.75
SIDE SALAD	3.35
HOT WINGS	3.55
MOZZARELLA STICKS	4.20
SAMPLER PLATE	5.80

~ SANDWICHES ~

BARBECUE	6.55
----------	------



Plan for the Day

- Pre-break course evaluations
- Design Class Diagrams
- Design exercise that should help with Homework #3



Help Me Help You

- Pre-break course evaluation on ANGEL
- Please take 10 minutes or so to help me improve the course





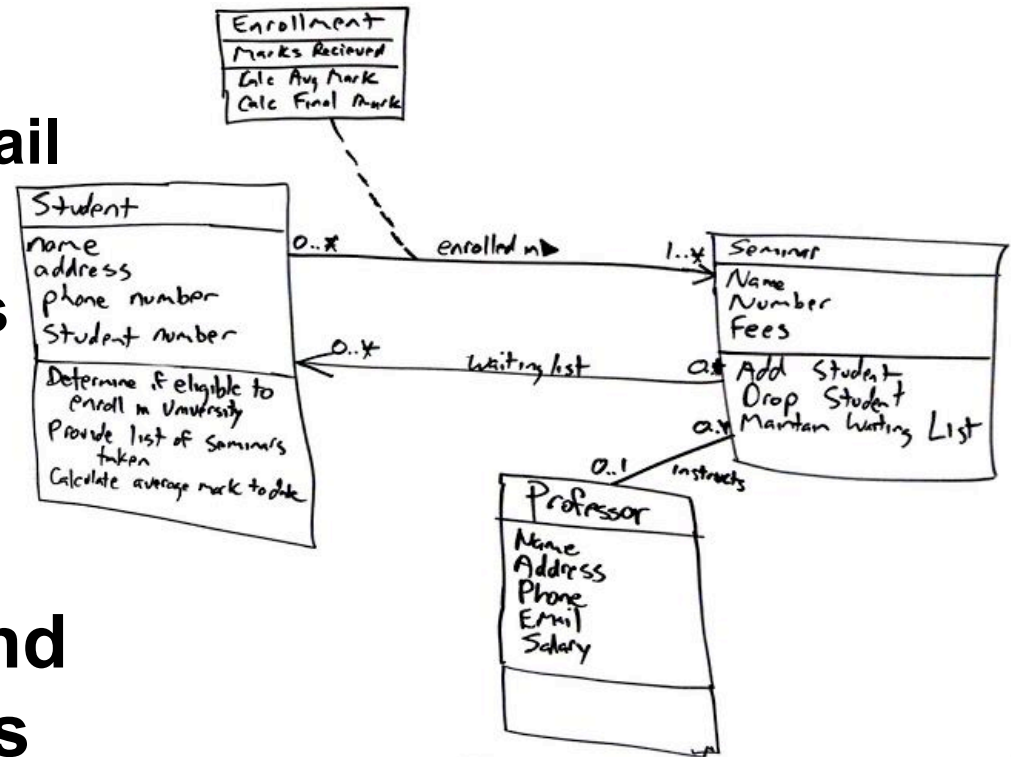
UML Class Diagrams

Design Class Diagrams (DCD) 1/2

■ Creation of DCDs

builds on prior:

- Domain Model (+detail to class definitions)
- Interaction diagrams (identifies class methods)

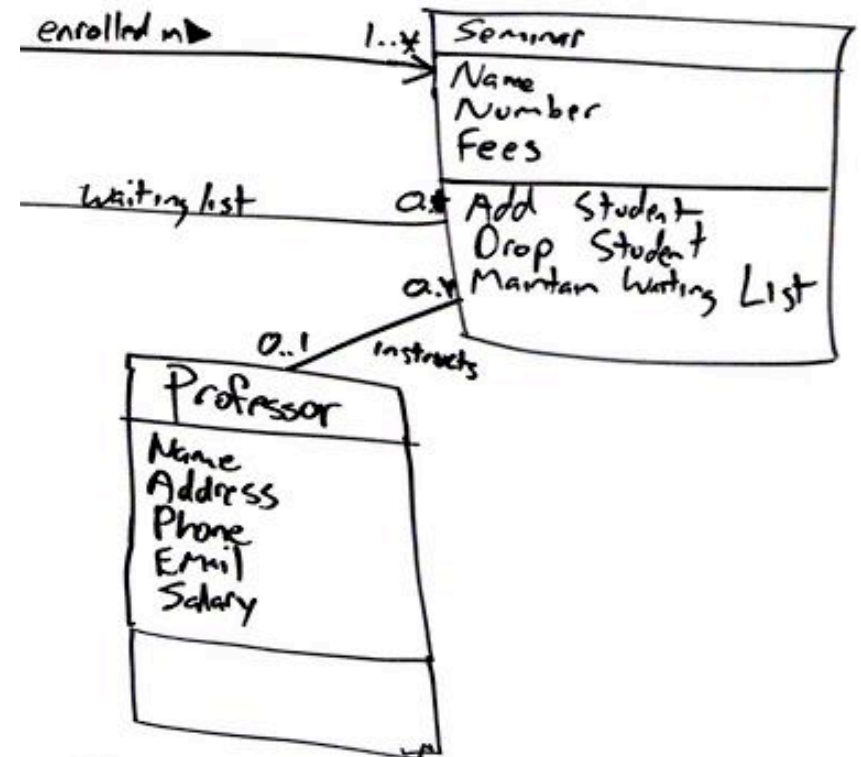


- Creation of DCDs and interaction diagrams are usually created in parallel

Design Class Diagrams (DCD) 2/2

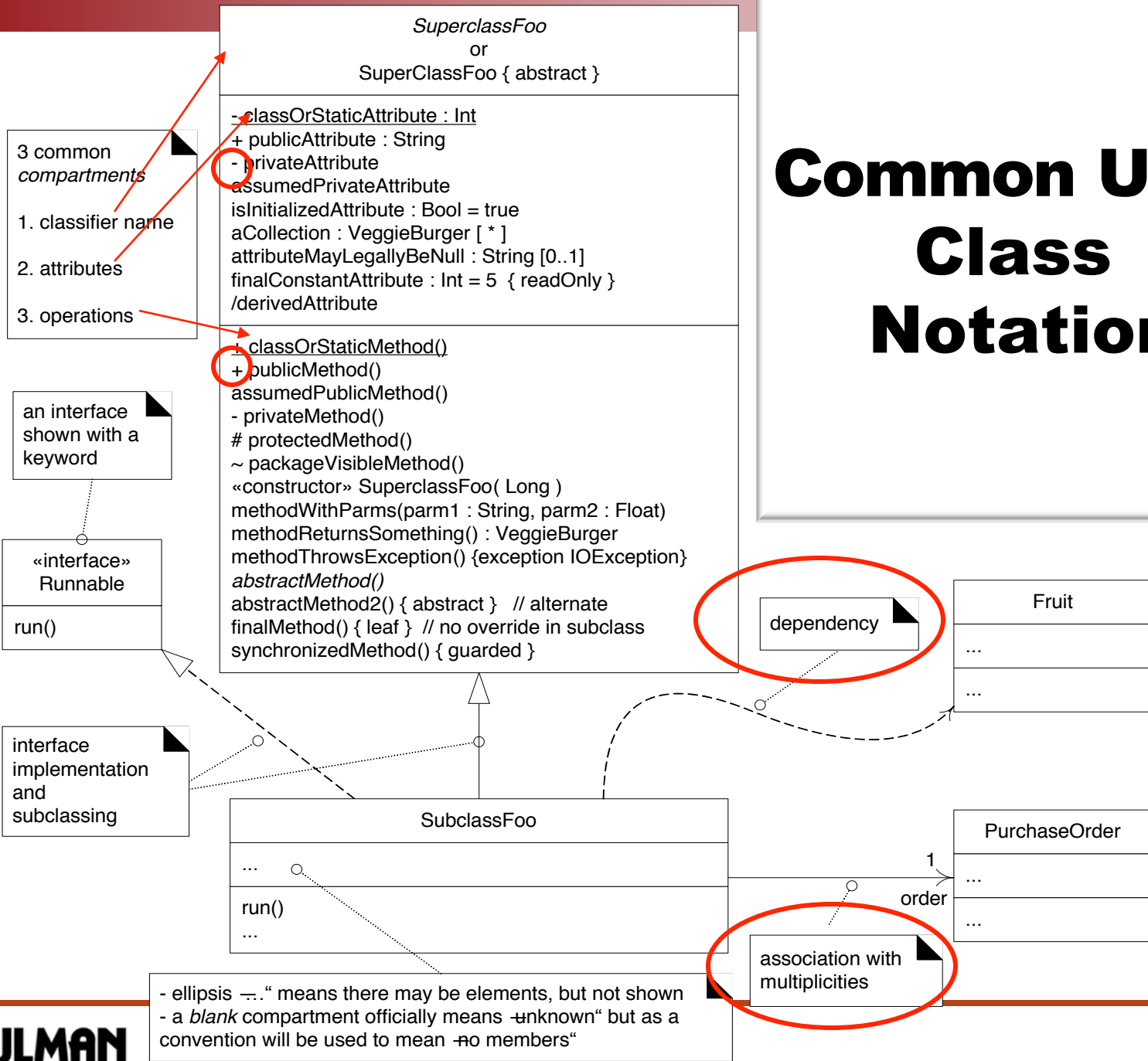
DCDs illustrates the specifications for software classes and interfaces including:

- Classes, associations, and attributes
- Interfaces, with their operations and constants
- Methods
- Attribute type information
- Navigability
- Dependencies





Common UML Class Notation



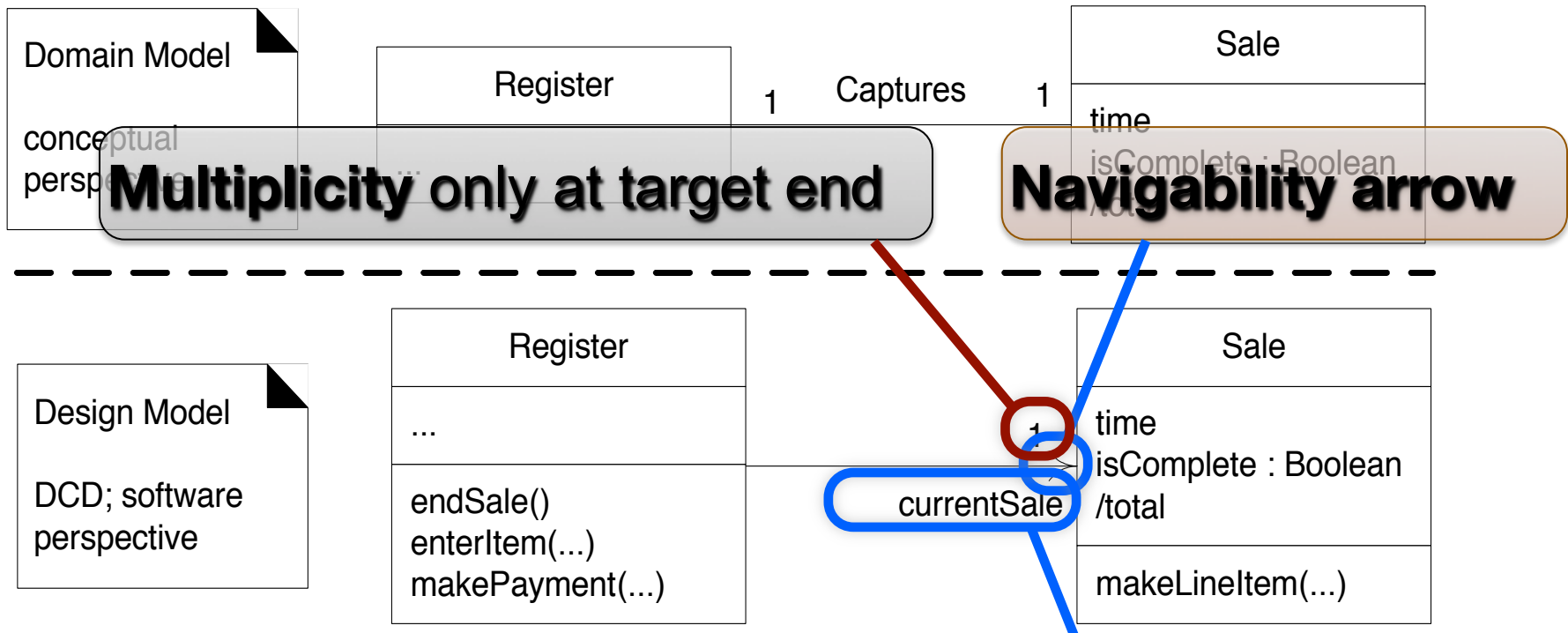
- ellipsis "... " means there may be elements, but not shown
- a blank compartment officially means "unknown" but as a convention will be used to mean "no members"



Recipe for a Design Class Diagram

- 1) Identify all the **classes** participating in the software solution by analyzing the interaction diagrams
- 2) Draw them in a class diagram
- 3) Duplicate the **attributes** from the associated concepts in the conceptual model
- 4) Add **method** names by analyzing the interaction diagrams
- 5) Add **type** information to the attributes and methods
- 6) Add the **associations** necessary to support the required attribute visibility
- 7) Add **navigability** arrows to the associations to indicate the direction of attribute visibility
- 8) Add **dependency** relationship lines to indicate non-attribute visibility

Class Diagrams Do Double Duty



- Call them *Domain Models* for analysis at the conceptual level
- Call them *Design Class Diagrams* for design

Attribute Text vs. Association Line Notation

using the attribute text notation to indicate Register has a reference to one Sale instance



Avoid

OBSERVE: this style *visually* emphasizes the connection between these classes



Role name is attribute name



Preferred

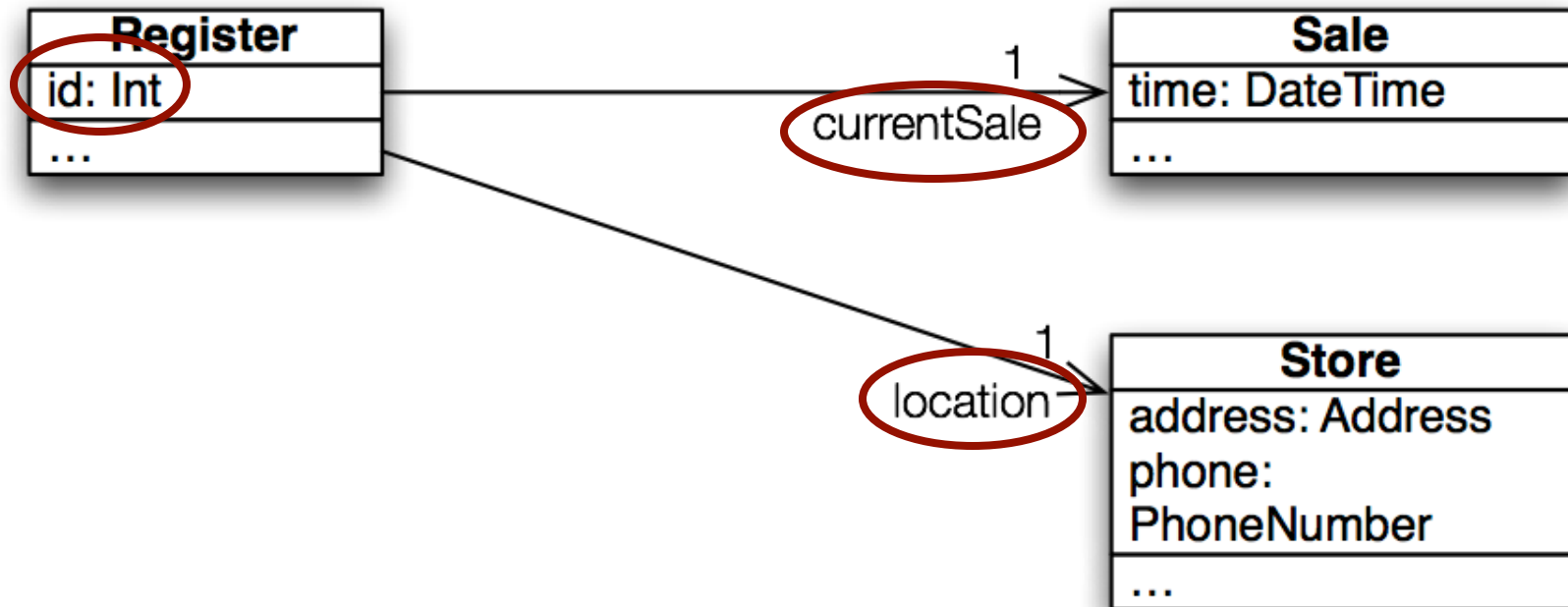
using the association notation to indicate Register has a reference to one Sale instance

thorough and unambiguous, but some people dislike the possible redundancy



Avoid

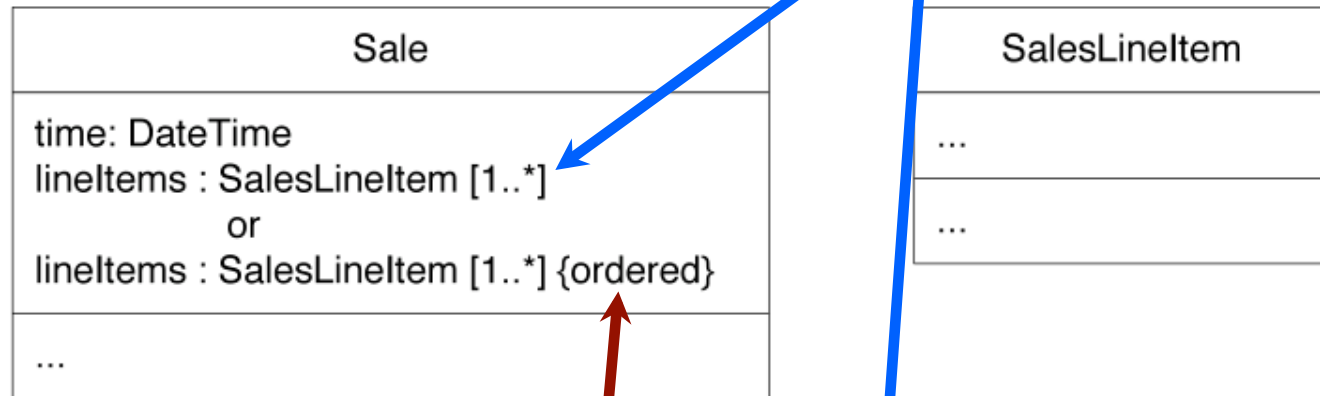
Guideline Good Practice: Example



Showing Collection Attributes

Multiplicities

1



2



Preferred, less visual clutter

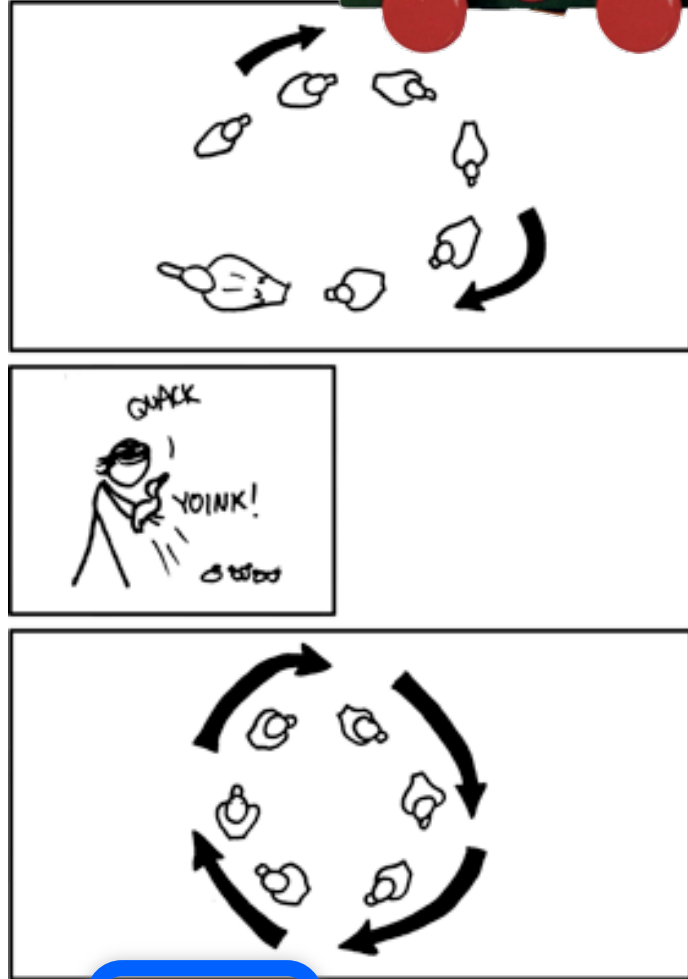
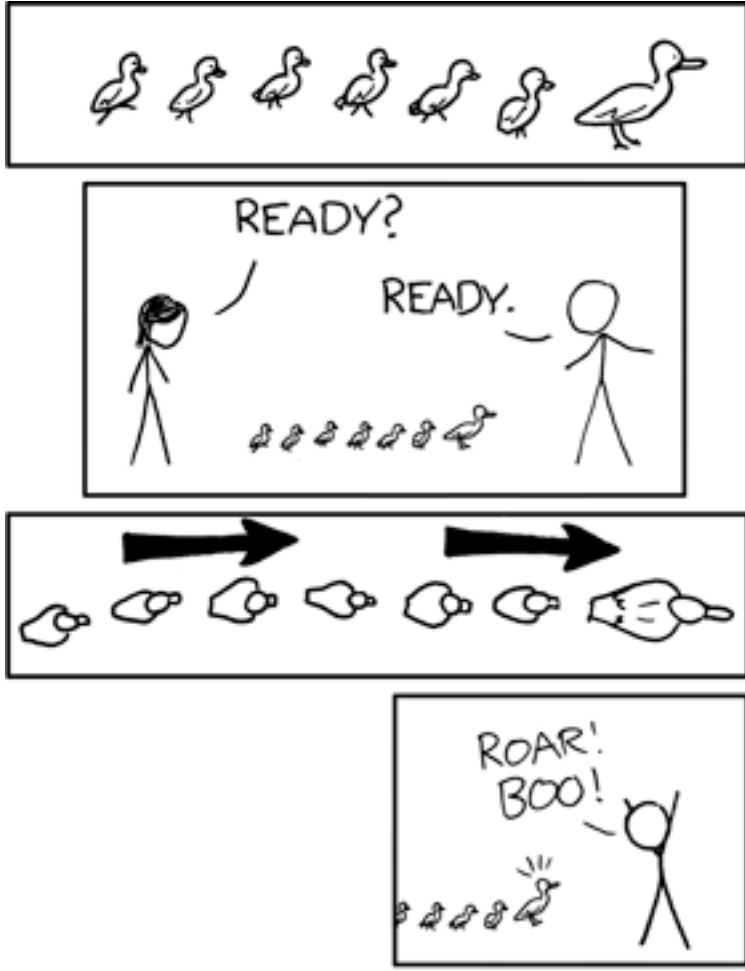
Constraints



Operations

- **Syntax:**
 - **visibility name (paramName:type, ...) : returnType**
{properties}
 - **+ getPlayer(name:String) : Player {exception IOException}**
- **Also use syntax of implementation language**
 - **public Player getPlayer(String name) throws IOException**
- ***Operation vs. operation contract vs. method***

Cartoon of the Day



OPERATION: DUCKLING LOOP

<http://www.brickfetish.com/toys/duck.html>

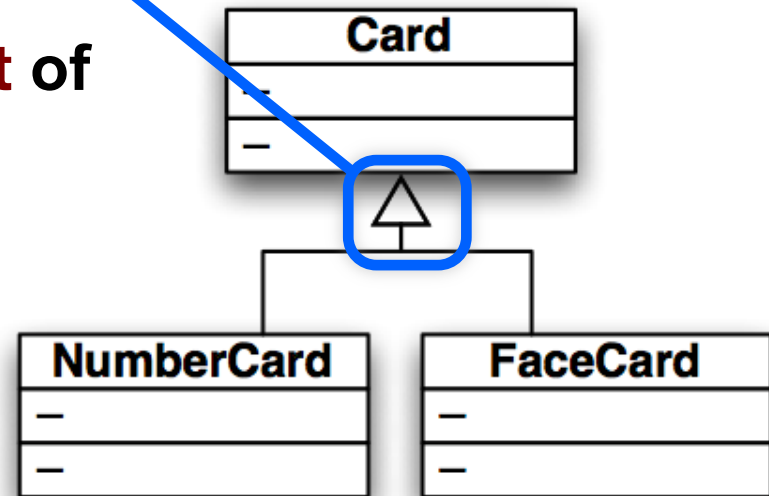


Keywords Categorize Model Elements

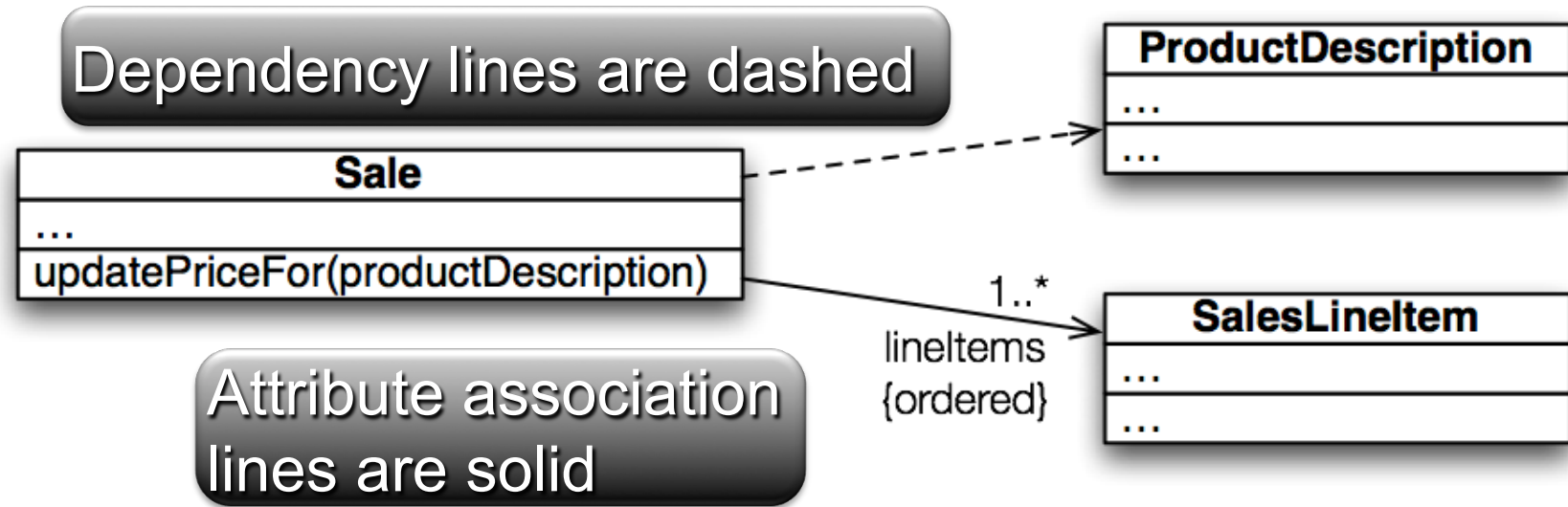
Keyword	Meaning	Example Usage
«actor»	classifier is an actor	shows that classifier is an actor without getting all xkcd 😊
«interface»	classifier is an interface	«interface» MouseListener
{abstract}	can't be instantiated	follows classifier or operation
{ordered}	set of objects has defined order	follows role name on target end of association
{leaf}	can't be extended or overridden	follows classifier or operation

Generalization

- In Domain Model:
 - Says that the set of all NumberCards is a **subset** of the set of all Cards
- In DCD:
 - Says that, and that NumberCard **inherits** from Card



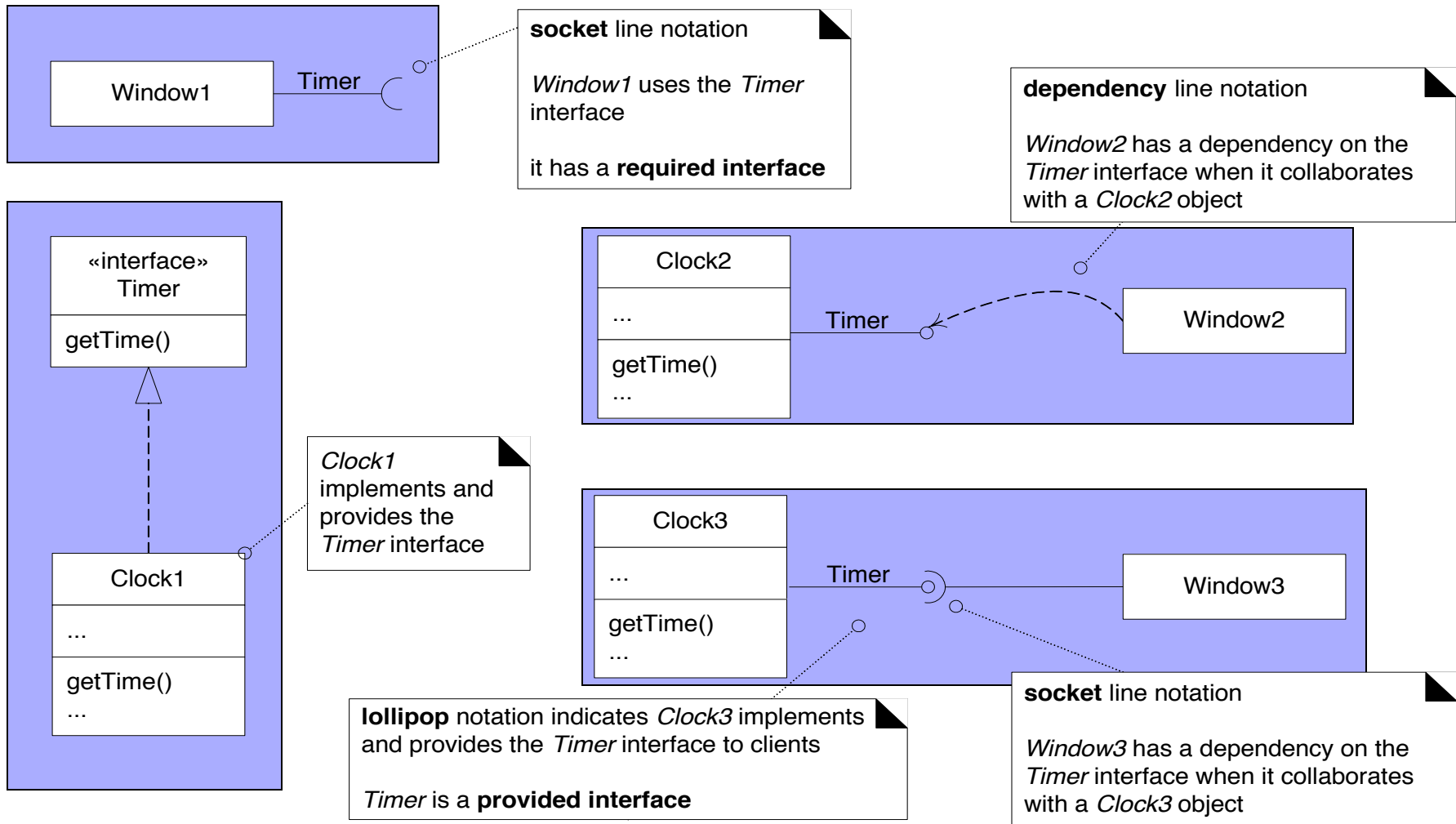
Dependencies



Use dependency lines when a more specific line type doesn't apply.

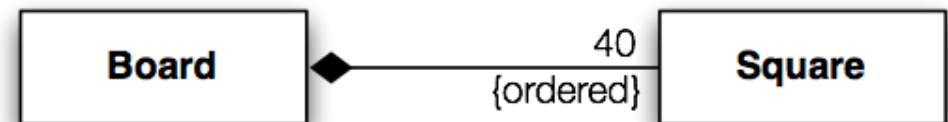
Can label dependency arrows:
e.g. «call», «create»

Interfaces in UML



Composition


- More powerful than an attribute arrow
- Describes “whole-part” relationship



- Implies

- Instance of part belongs to only one composite at a time
- Part always belongs to a composite
- Composite creates/deletes parts

Association name in composition is always implicitly some “has-part” relation. So, it’s common to omit association or role name with compositions



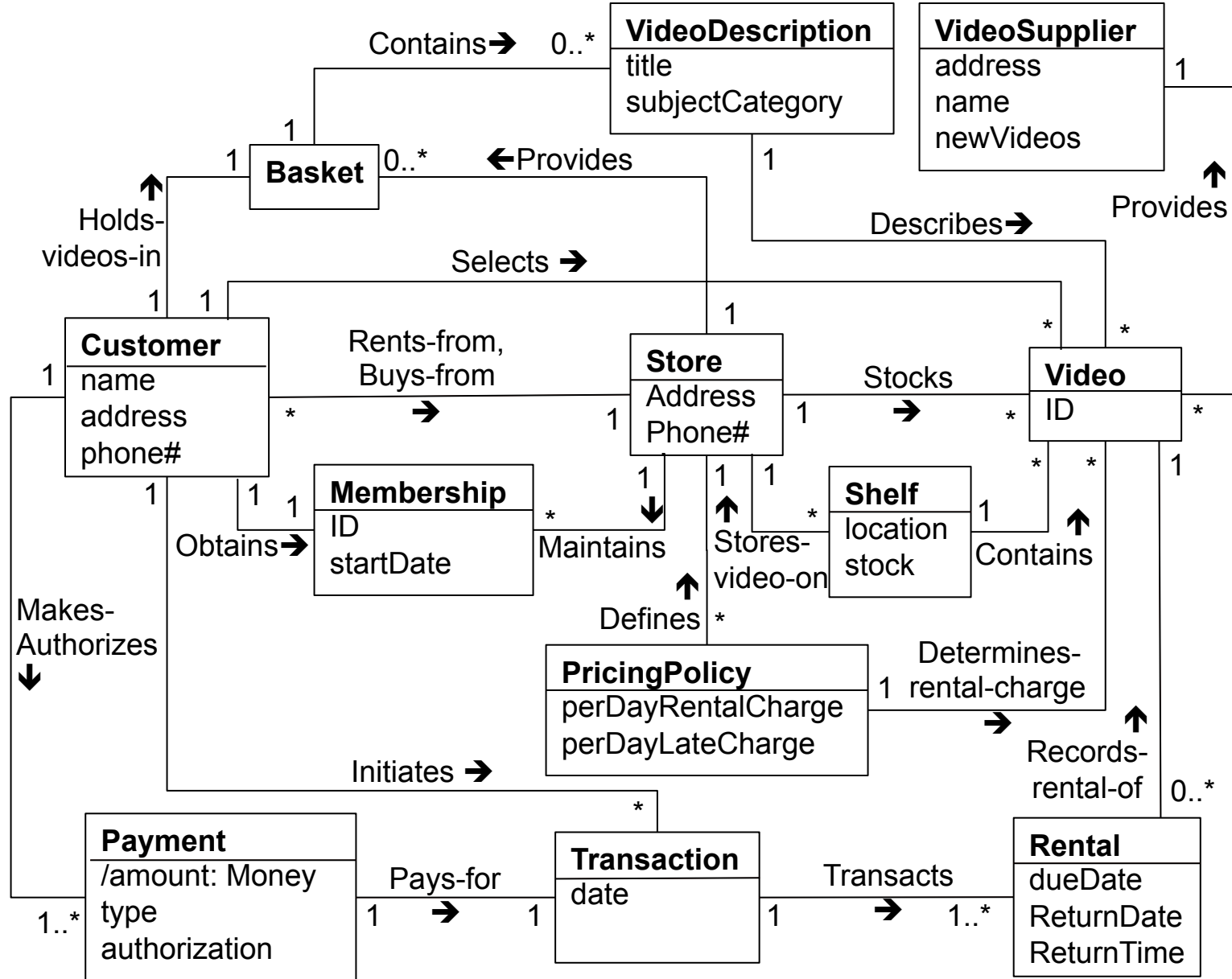
Interaction Diagrams and Class Diagrams

- Interaction diagrams show *dynamic* behavior
- Class diagrams show *static* behavior
- Tips:
 - Draw concurrently
 - Use two adjacent whiteboards, one for static and one for dynamic
 - Sketch communication diagrams, document using sequence diagrams

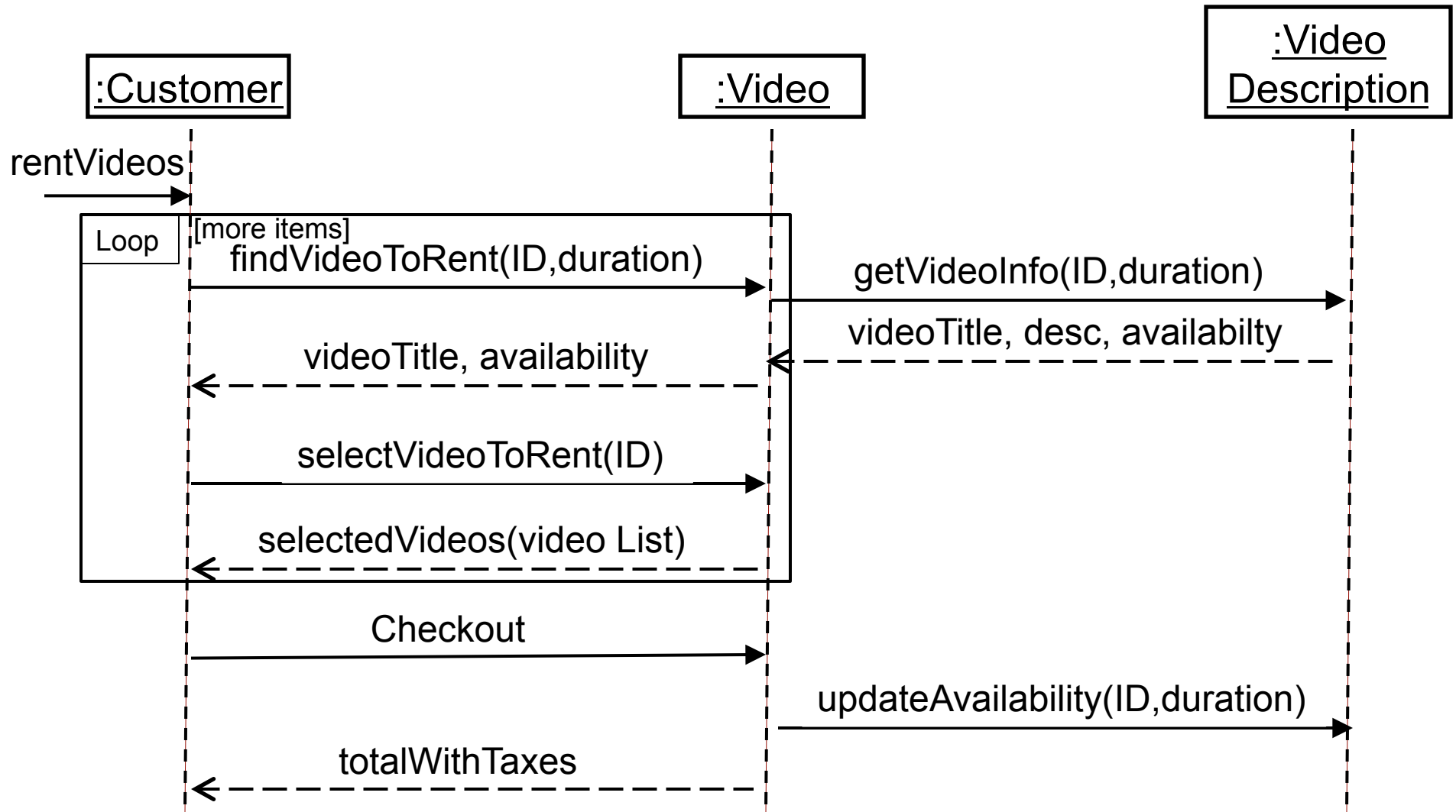
Exercise on Sequence Diagrams

- Break up into your project teams
- Given the following:
 - Domain Model and a Sequence Diagram
- Draw Design Class Diagram showing a Customer and the classes in a Rental package and a Video package for BrickBusters Video Store. Try to minimize dependencies.





An SD Solution for Rent Video Example





Homework and Milestone Reminders

- Read Chapter 17 on Responsibility Driven Design
- Homework 3 – BBVS Logical Architecture and Preliminary Design
 - Due by 5:00pm on Tuesday, January 4th, 2011
- Milestone 3 – Junior Project SSDs, OCs, and Logical Architecture
 - Due by 11:59pm on Friday, January 7th, 2010
- 5% extra credit on Milestone 3 and Homework3 if you finish by 11:59pm, Friday before break!

A CD Solution for Rent Video Example

