Object-Oriented Design Examples & Exam Review

Shawn Bohner

Office: Moench Room F212

Phone: (812) 877-8685

Email: bohner@rose-hulman.edu

Agenda

- Work some O-O Design Examples
- Outline Thursday’s Exam
- Review what we covered so far

Example: Grading System Problem Statement

The system will help instructors and teaching assistants provide thorough, timely feedback to students on assignments. The system will make grading more efficient, allowing students to more quickly receive feedback and course staff to devote more time to improving instruction.

The system will take a collection of student solutions to an assignment as PDF files or some other convenient, open standard. It will allow the grader to “write” feedback on student submissions. It will keep track of the grader's place in each assignment so that he or she can grade every student's answer to question 1, then question 2, and so on. Finally the application will create new PDF files including comments for return to the students.

Besides feedback, the system will help with calculating grades. The grader can associate points with each piece of feedback, so that the application can calculate points earned on the assignment. The grader will be able to drag remarks from a “well” of previous feedback to give the same feedback to multiple students (and deduct or add the same number of points). The points associated with a particular piece of feedback can be edited, causing the system to update the score calculations for every student that received that feedback.

A Sampling of Use Cases

- Create assignment
- Import student submissions
- Create feedback item
- Edit feedback item
- Add feedback to a submission
- Export graded student submissions

See Domain Model and SSDs in handout

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Create New Assignment

Operation	*createNewAssignment(title, description, dueDate, authors)*
Cross References	Use Case: Create Assignment
Preconditions	none
Postconditions	▪ an *Assignment* instance, *assignment*, was created ▪ the attributes of *assignment* were set from the corresponding arguments ▪ a list, *instructors*, of new *Instructor* instances were created for each *author* in *authors* ▪ for each *instructor* in *instructors*, *instructor.name* was set to the corresponding *author* in *authors* • *assignment* was associated with *instructors*

Q2,4

Create New Rubric

Operation	*createNewRubric(assignment, pointsAvailable, initialRequirements, authors)*
Cross References	Use Case: Create Assignment
Preconditions	*assignment* is an existing *Assignment* in the system
Postconditions	▪ a *Rubric* instance, *rubric*, was createdthe attributes of *rubric* were set from the corresponding argumentsa list, *instructors*, of new *Instructor* instances was created for each *author* in *authors*for each *instructor* in *instructors*, *instructor.name* was set to the corresponding *author* in *authorsrubric* was associated with *instructorsrubric* was associated with *assignment*

Q4 cont.

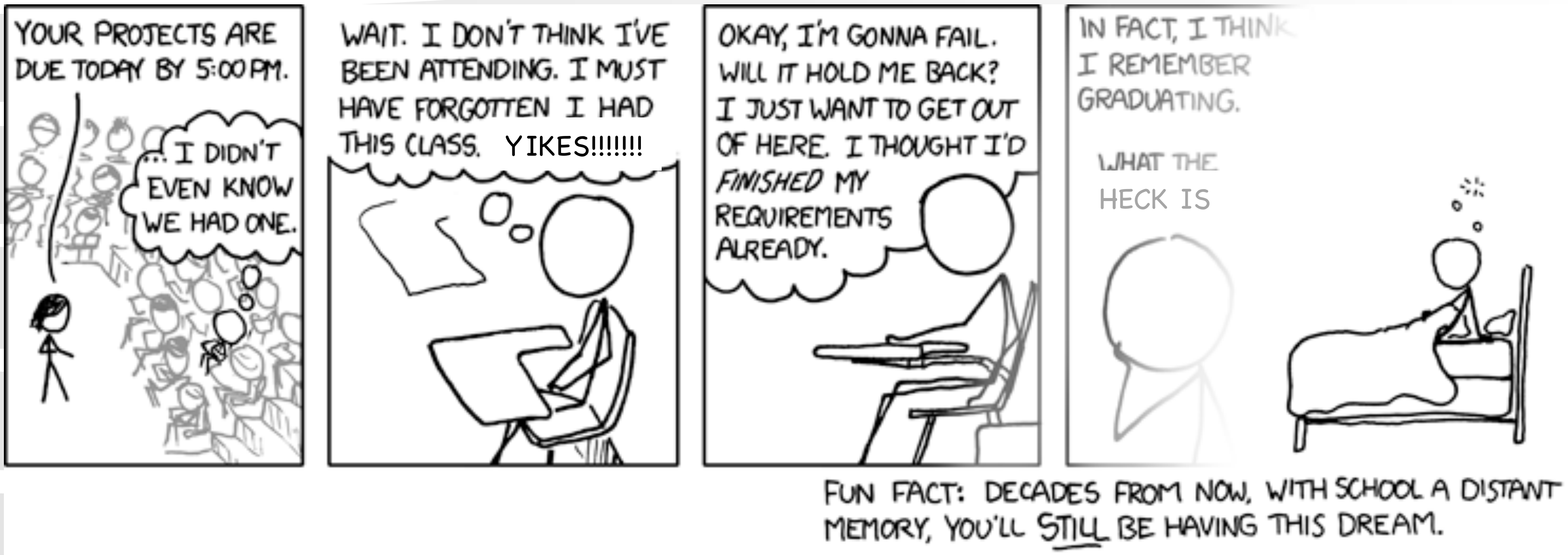
Add Requirement

Operation	*addRequirement(rubric, requirement)*
Cross References	Use Case: Create Assignment
Preconditions	*rubric* is an existing *Rubric* in the system
Postconditions	▪ *requirement* was appended to *rubric.requirements*

Q4 cont.

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Students


YOUR PROJECTS ARE DUE TODAY BY 5:00PM.
...I DIDN'T EVEN KNOW WE HAD ONE.
WAIT. I DON'T THINK I'VE BEEN ATTENDING. I MUST HAVE FORGOTTEN I HAD THIS CLASS.
YIKES!!!!!!!
OKAY, I'M GONNA FAIL. WILL IT HOLD ME BACK? I JUST WANT TO GET OUT OF HERE. I THOUGHT I'D FINISHED MY REQUIREMENTS ALREADY.
IN FACT, I THINK I REMEMBER GRADUATING.
WHAT THE HECK IS
FUN FACT: DECADES FROM NOW, WITH SCHOOL A DISTANT MEMORY, YOU'LL STILL BE HAVING THIS DREAM.

Hopefully NOT what you feel like today…
Homework #5 Due by 5:00pm

Note: Aren't you glad you are not at a large school where you are one of 50-100 students in a class

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Edit Feedback Item

Operation	*editFeedbackItem(item, title, points, comments)*
Cross References	Use Case: Edit Feedback Item
Preconditions	*item* is an existing *FeedbackItem* in the system
Postconditions	▪ the attributes of *item* were updated based on the other arguments

Q4 cont.

Add Submission

Operation	*addSubmission(assignment, studentName, submissionData, submissionDate)*
Cross References	Use Case: Import Student Submissions
Preconditions	*assignment* is an existing *Assignment* in the system
Postconditions	▪ a new *Submission* instance, *submission*, was created*submission.studentAnswers* was set to *submissionDatasubmission.submissionDate* was set to *submissionDatesubmission* was associated with *assignment*a new *Student* instance, *student*, was created*student.name* was set to *studentNamesubmission* was associated with *student*

Q3,4

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Progression From Analysis into Design

- Use Cases drove the development of
  - Domain Model (DM)
  - System Sequence Diagrams (SSD)
  - Operation Contracts (OC)
- DM is starting point for Design Class Diagram
- SSDs help identify system operations, the starting point for Interaction Diagrams
  - System operations are the starting messages
  - Starting messages are directed at controller objects
- Use OC post-conditions to help determine…
  - What should happen in the interaction diagrams
  - What classes belong in the design class diagram

Thursday's Exam

- Basic structure
  - 10-15 minutes of breadth (multiple choice and short answer)
  - Staged problem solving
    - Finish first part, hand it in to get next part
    - Next part has our answer to first part for you to use on second part
    - And so on…

- Exam is 15% of course grade

Engineering Design – A Simple Definition

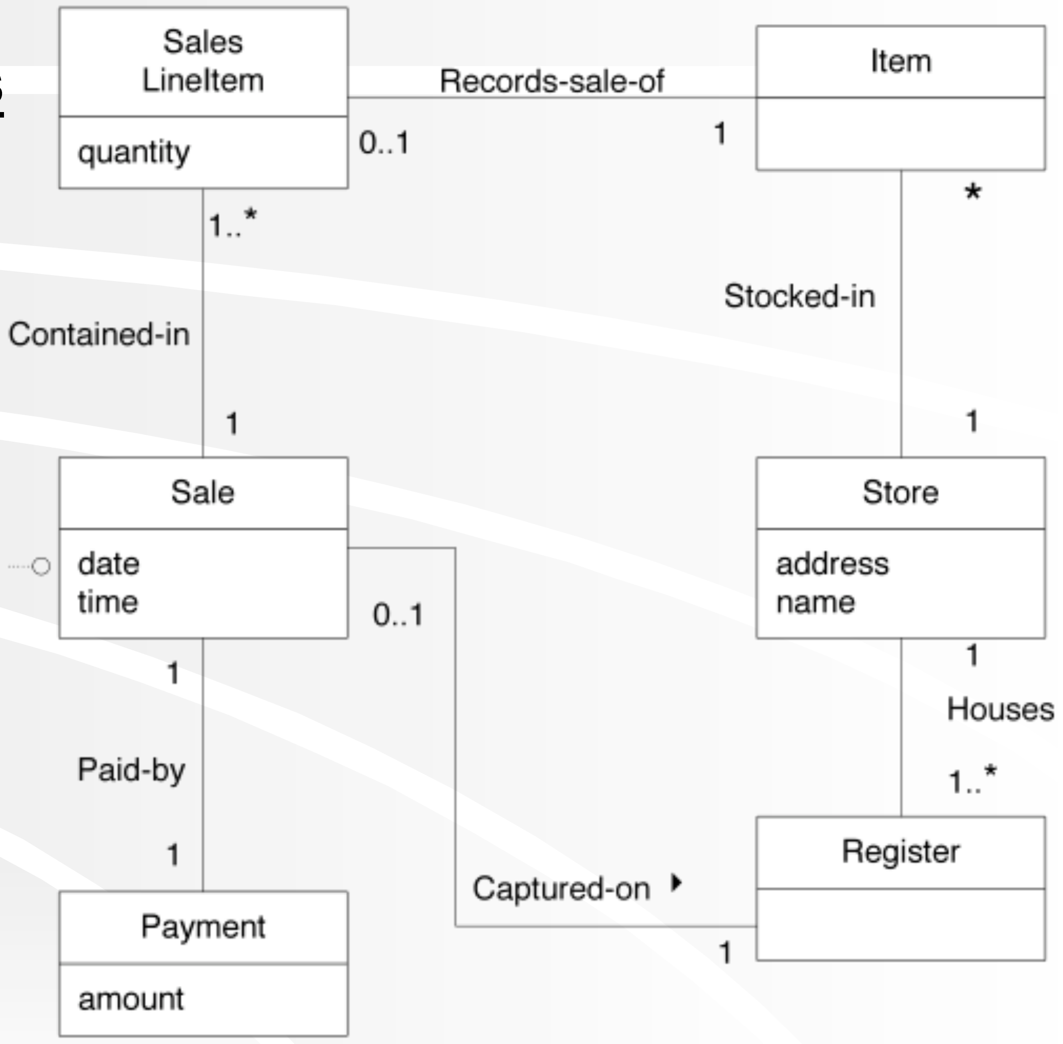
- “Design” specifies the strategy of “how” the Requirements will be implemented
- Design is both a “Process” … and an “Artifact”

Ways to use Unified Modeling Language (UML)

- Sketch
- Blueprint
- Executable programming language

Domain Model – An Abstraction of Conceptual Classes

- Most important model in Object-Oriented Analysis
- Illustrates noteworthy concepts in a domain
- Source of inspiration for designing software objects
- Goal: to lower representational gap
- Helps us understand & maintain the software



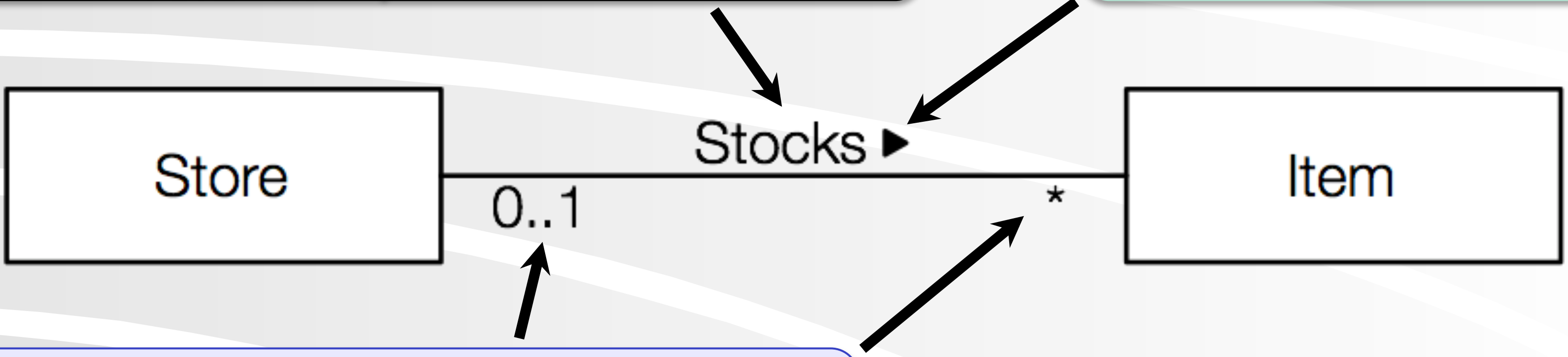
Strategies to Find Conceptual Classes

1. Reuse or modify existing models
2. Identify noun phrases; linguistic analysis
3. Use a category list

Associations

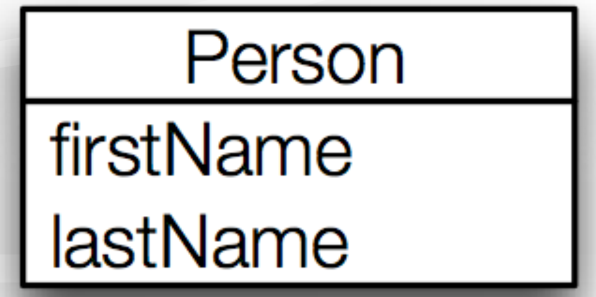
Association name:
- ✓ Use verb phrase
- ✓ Capitalize
- ✓ Typically camel-case or hyphenated
- ✓ Avoid “has”, “use

Reading direction: Can exclude if association reads left-to-right or top-to-bottom



Multiplicity (Cardinality):
- ✓ ‘*’ means “many”
- ✓ x..y means from x to y inclusively

Attributes



- Include attributes that the requirements suggest need to be remembered
- The usual ‘primitive’ data types
- Common compound data types
- Notation (“[ ]”indicate optional parts):
  - [+|-] [/] *name* [: [*type*] [*multiplicity*]] [= *default*] [{*property*}]

Visibility

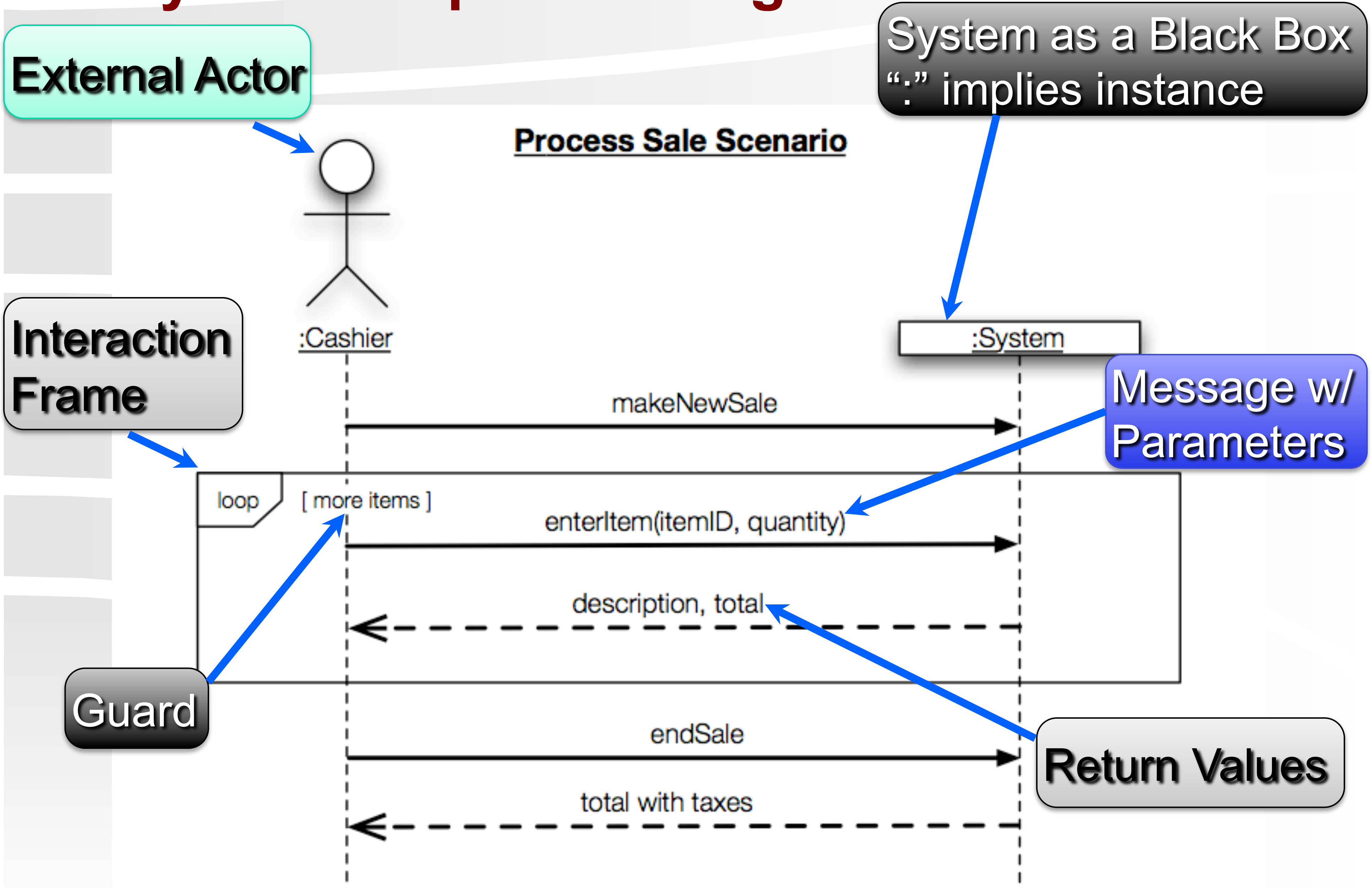
Derived

e.g., readOnly

Summary of Domain Model Guidelines

- Classes first, then associations and attributes
- Use existing models, category lists, noun phrases
- Include “report objects”, like Receipt, if they’re part of the business rules
- Use terms from the domain
- Don’t send an attribute to do a conceptual class’s job
- Use description classes to remember information independent of instances and to reduce redundancy
- Use association for relationship that must be remembered
- Be parsimonious with associations
- Name associations with verb phrases (not “has” or “uses”)
- Use common association lists
- Use attributes for information that must be remembered
- Use data type attributes
- Define new data types for complex data
- Communicate with stakeholders

System Sequence Diagrams



ROSE-HULMAN INSTITUTE OF TECHNOLOGY

How To "Tips" on Creating SSDs

- Show one scenario of a use case
- Show events as intentions, not physical implementation
  - E.g., *enterItem* not *scanItem*
- Start system event names with verbs
- Can model collaborations between systems

Parts of the Operation Contract

Operation: Name Of operation, and parameters.

Cross-References: (*optional*) Use cases this can occur within.

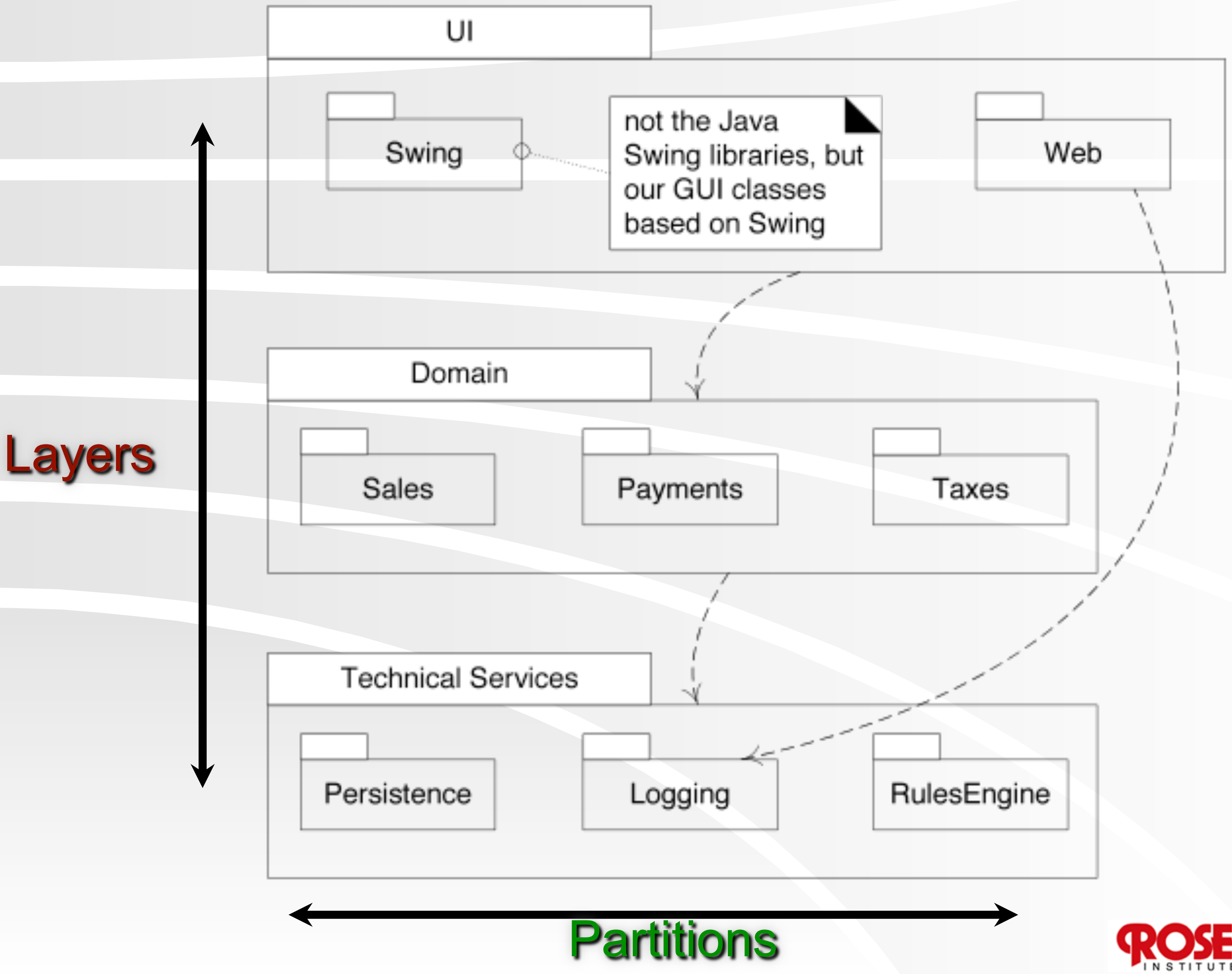
Preconditions: Noteworthy assumptions about the state of the system or objects in the Domain Model before execution of the operation.

Postconditions: The state of objects in the Domain Model after completion of the operation.

Postconditions

- Describe changes in the state of DM objects
- Typical changes: Created/Deleted Instances, Formed/Broke Associations, Changed Attributes
- Express post-conditions in the past tense
- Give names to instances
- Capture information from system operation by noting changes to domain objects

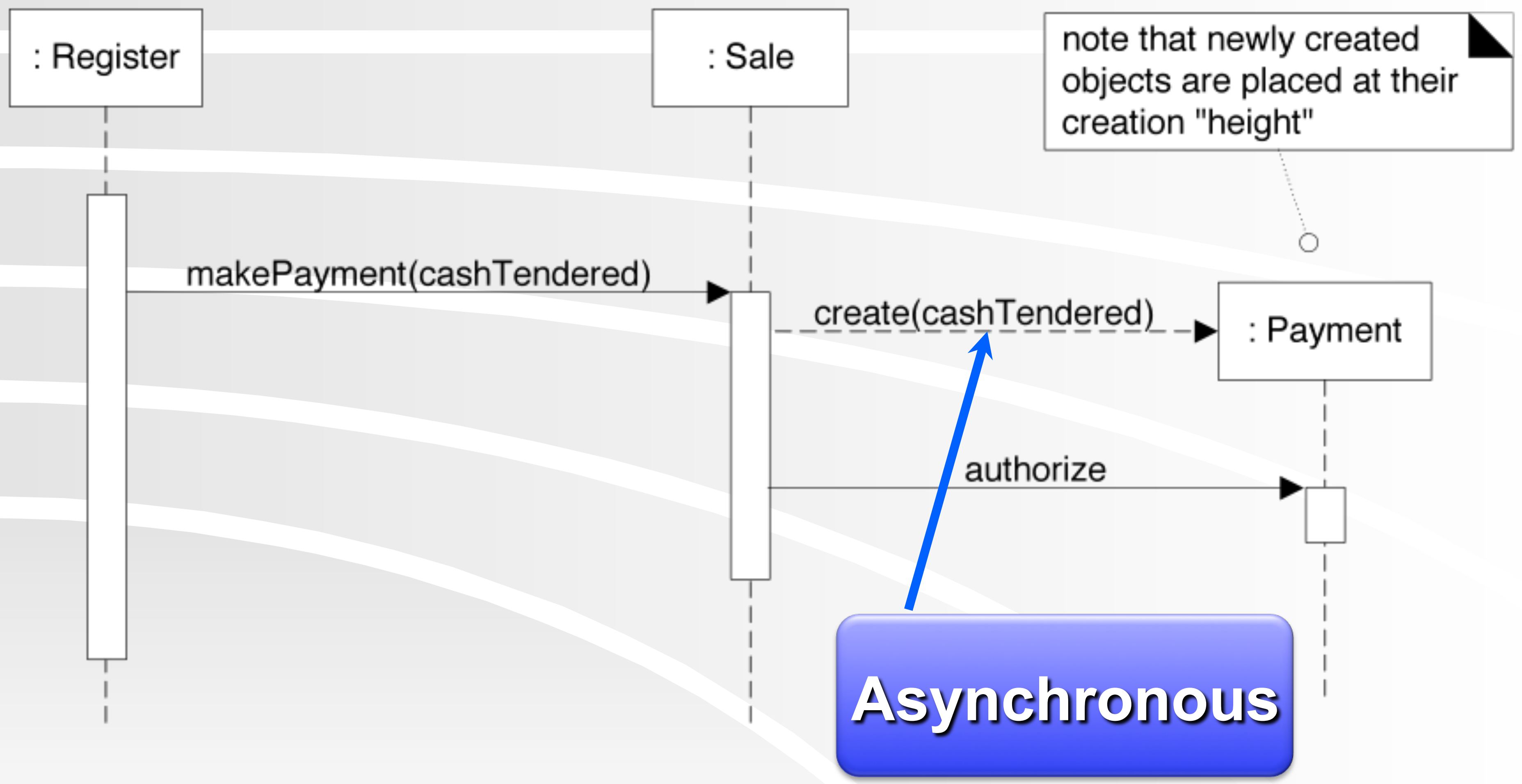
Logical Architecture



Dynamic Modeling with Interaction Diagrams

- Sequence Diagrams (SD)
  - Clearer notation and semantics
  - Better tool support
  - Easier to follow
  - Excellent for documents
- Communication Diagrams (CD)
  - Much more space efficient
  - Easier to modify quickly
  - Excellent for UML as sketch

Sequence Diagrams

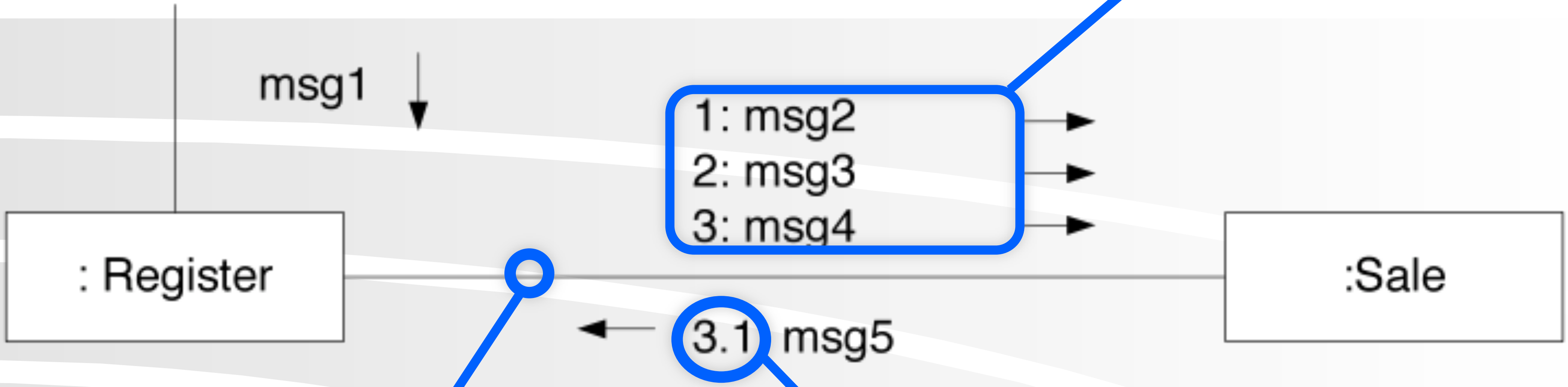

: Register
: Sale
note that newly created objects are placed at their creation "height"
makePayment(cashTendered)
create(cashTendered)
: Payment
authorize
Asynchronous

Common Frame Operators

Operator	Meaning
alt	"alternative", if-then-else or switch
loop	loop while guard is true, or *loop(n)* times
opt	optional fragment executes if guard is true
par	parallel fragments
region	critical region (single threaded)
ref	a "call" to another sequence diagram
sd	a sequence diagram that can be "called"

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Communication Diagrams

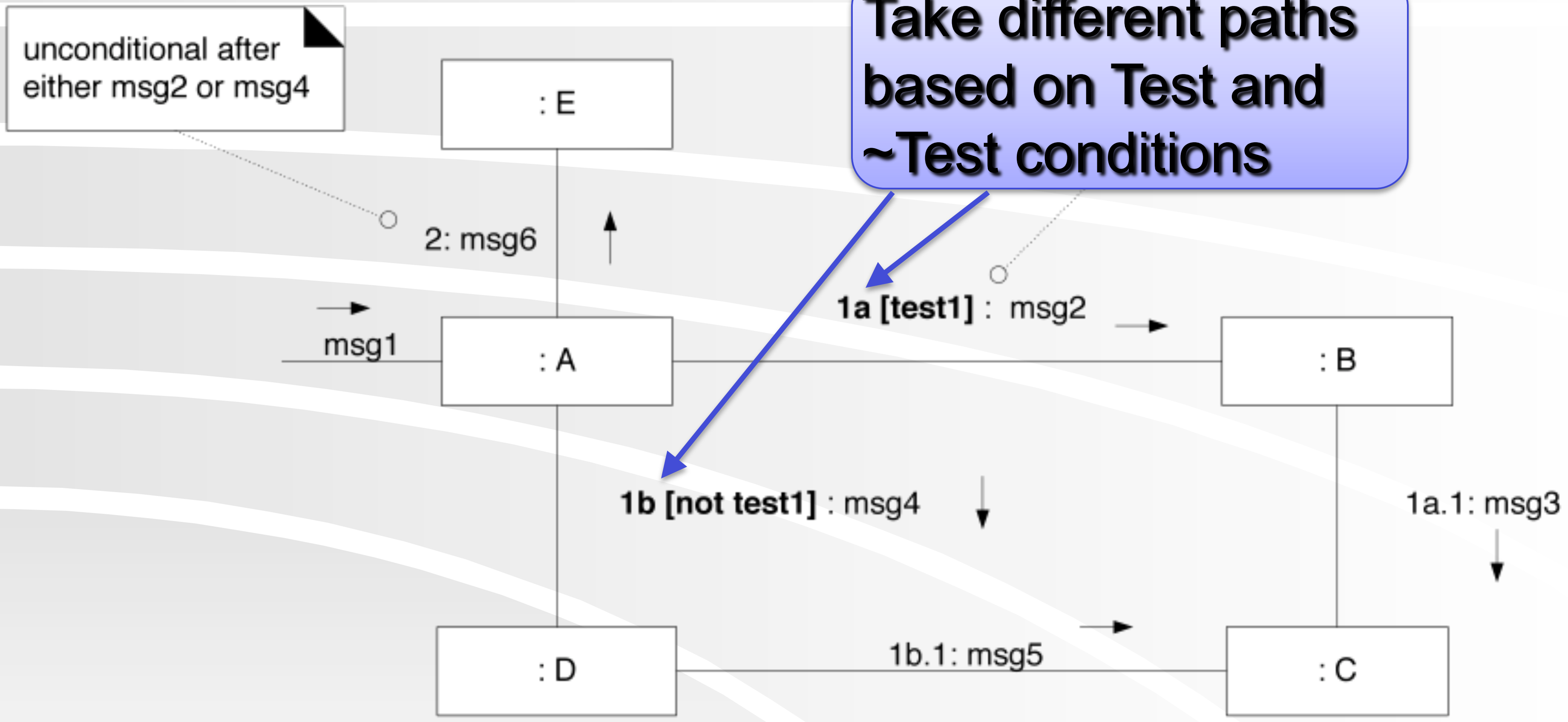


Multiple **messages** traverse links

Single **link** connects two objects

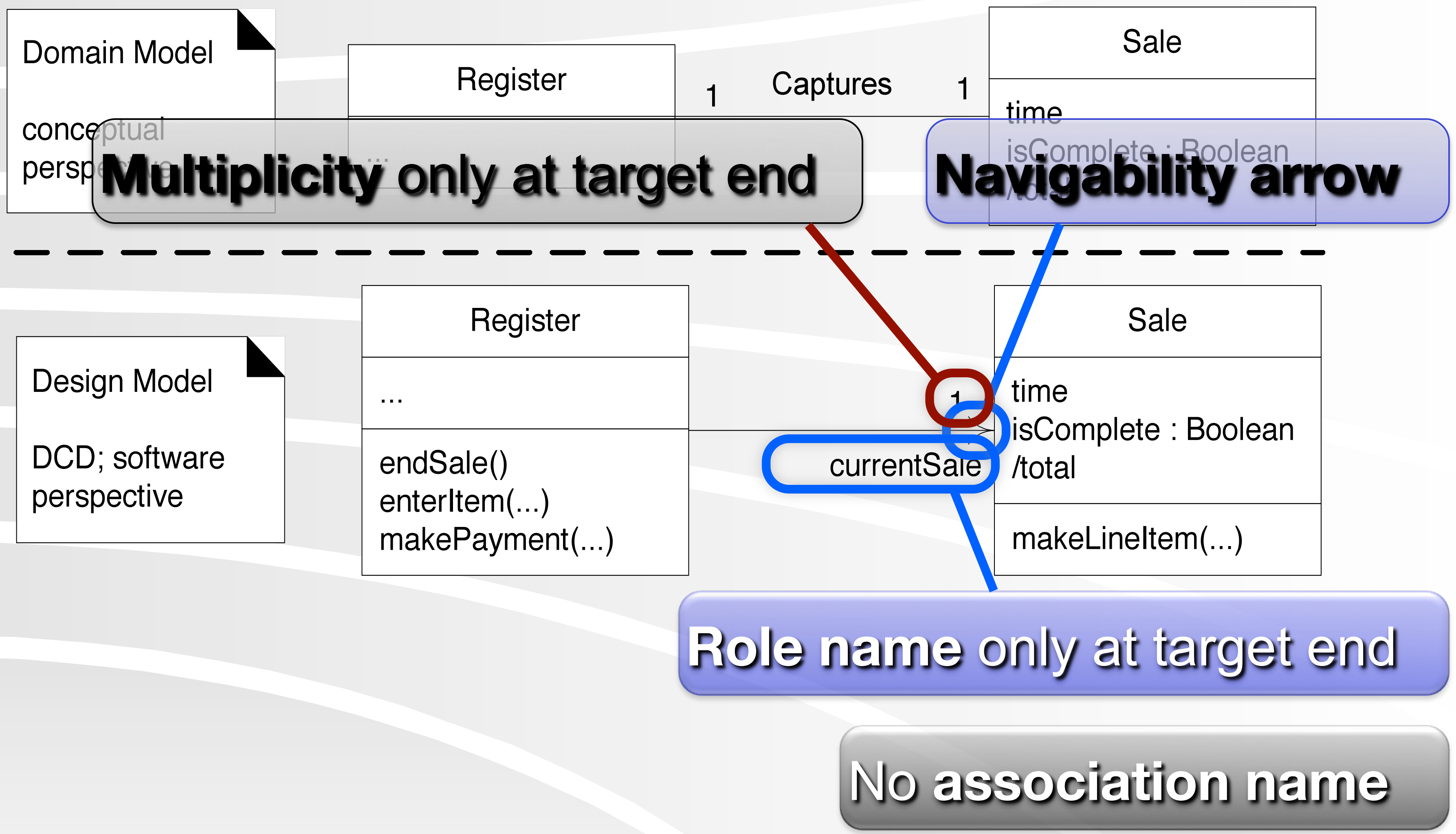
Sequence number gives ordering

Conditional Messages Use Guards



Take different paths based on Test and ~Test conditions

DMs to Design Class Diagrams



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

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

GRASP

- GRASP: General Responsibility Assignment Software Patterns (or Principles)
  - A set of patterns for assigning responsibilities to software objects
- Five Initial GRASPs
  1. Creator
  2. Information Expert
  3. Low Coupling
  4. Controller
  5. High Cohesion
- Four Later In Chapter 25
  - Polymorphism Pure Fabrication
    Indirection Protected Variations

RDD: Knowing and Doing Responsibilities

- “Doing” Responsibilities
  - Create another object
  - Perform a calculation
  - Initiate an action in an object
  - Control/coordinate activities of objects

- “Knowing” Responsibilities
  - Knowing it’s own encapsulated data
  - Knowing about other objects
  - Knowing things it can derive or calculate

Coupling

An Evaluative Principle

- A measure of how strongly one element:
  - is connected to,
  - has knowledge of, or
  - relies on other elements
- Want low (or weak) coupling

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Cohesion

An Evaluative Principle

- A measure of how strongly related and focused the responsibilities of a class (or method or package…) are
- Want high cohesion

Information Expert

- Problem: What is a general principle of assigning responsibilities?
- Solution: Assign a responsibility to the class that has the necessary information

Creator

- Problem: Who should be responsible for creating a new instance of some class?
- Solution: Make *B* responsible for creating *A* if…
  - *B* contains or is a composition of *A*
  - *B* records *A*
  - *B* closely uses *A*
  - *B* has the data to initialize *A*

The more matches the better.

Controller

- Problem: What first object beyond the UI layer receives and coordinates a *system operation*

- Solution: Assign the responsibility to either…
  - A façade controller, representing the overall system and handling all system operations, or
  - A use case controller, that handles all system events for a single use case

Homework and Milestone Reminders

- Read Chapter 20 for Monday
- Study for Exam on Thursday
- Homework 5 – Practice GRASP on Video Store Design and Midcourse Team Evaluation
  - Due by 5:00pm Tuesday, January 12th, 2010
  - NO LATE DAYS on this assignment