CSSE 374: More Object-Oriented Design Exercise and Exam Review

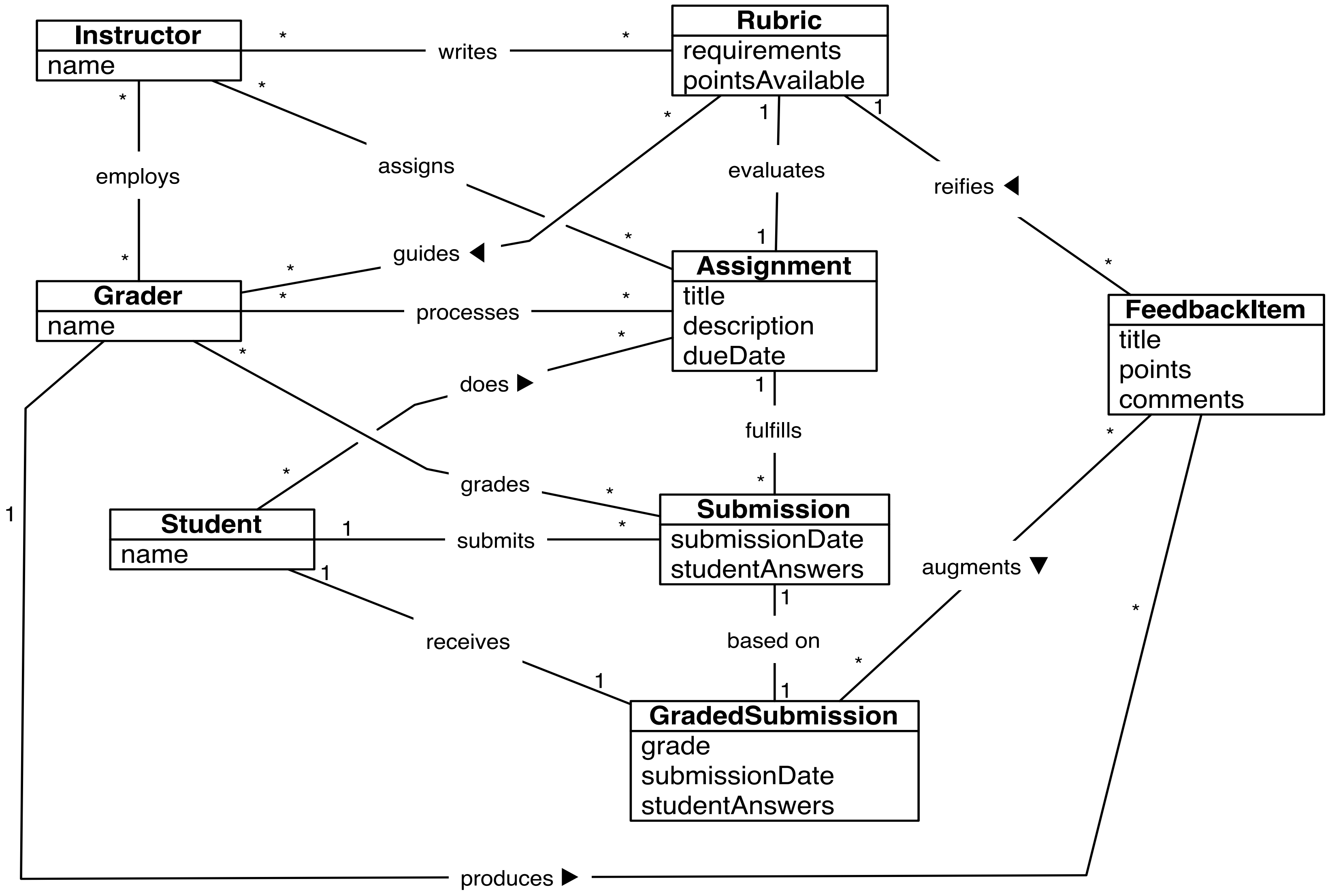
Shawn Bohner

Office: Moench Room F212

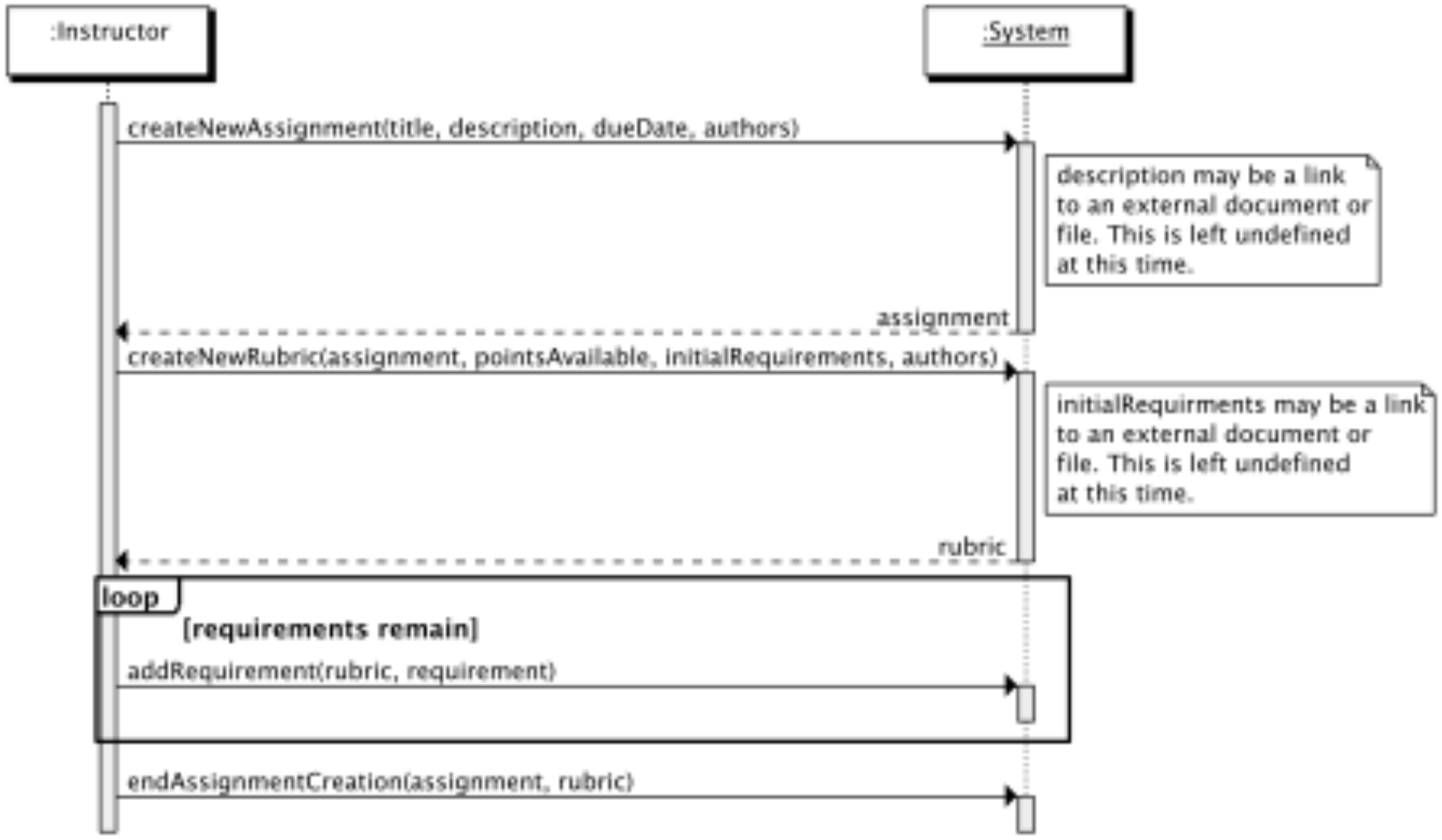
Phone: (812) 877-8685

Email: bohner@rose-hulman.edu

Domain Model for Grading System



Create Assignment Scenario


:Instructor
:System
createNewAssignment(title, description, dueDate, authors)
description may be a link to an external document or file. This is left undefined at this time.
assignment
createNewRubric(assignment, pointsAvailable, initialRequirements, authors)
initialRequirments may be a link to an external document or file. This is left undefined at this time.
rubric
loop
[requirements remain]
addRequirement(rubric, requirement)
endAssignmentCreation(assignment, rubric)

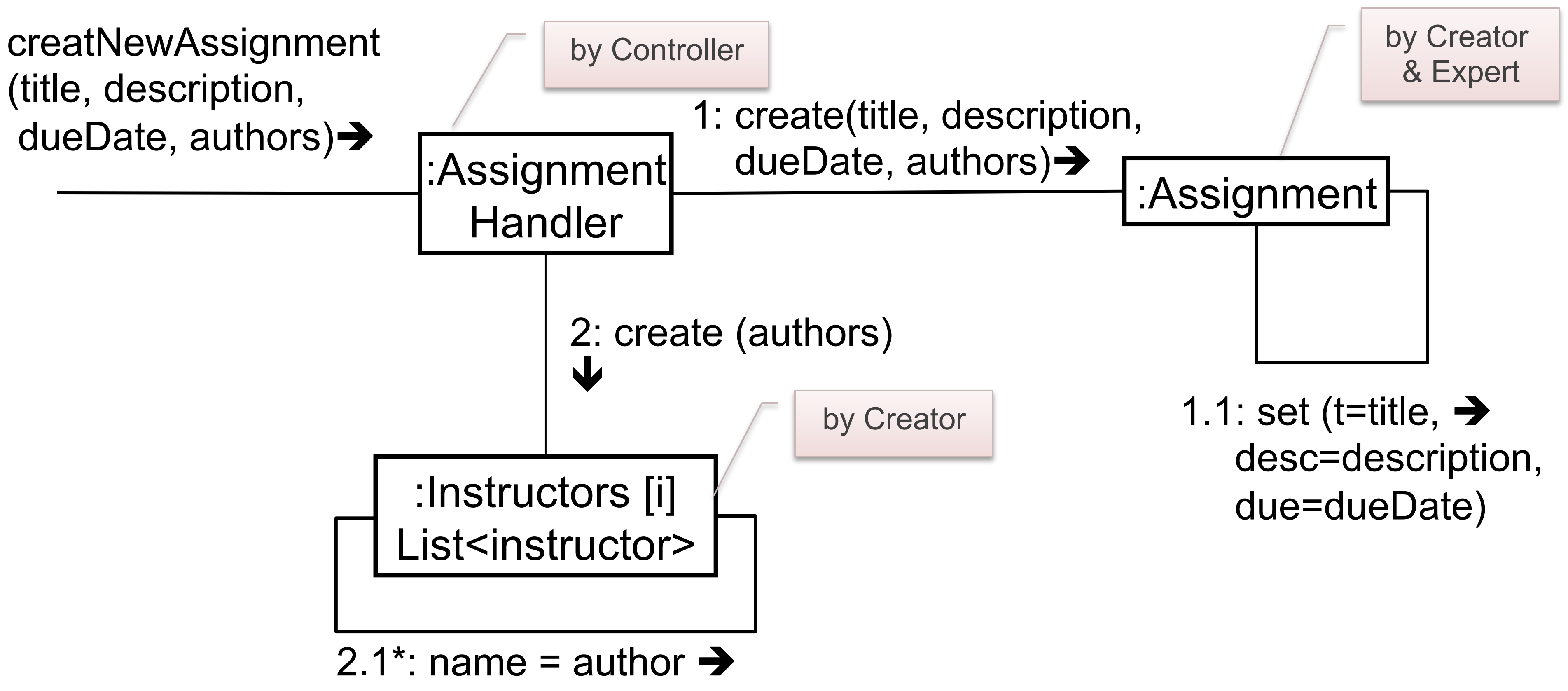
Create New Assignment

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 was created ▪ for each *instructor* in *instructors*, *instructor.name* was set to the corresponding *author* in *authors* • *assignment* was associated with *instructors*

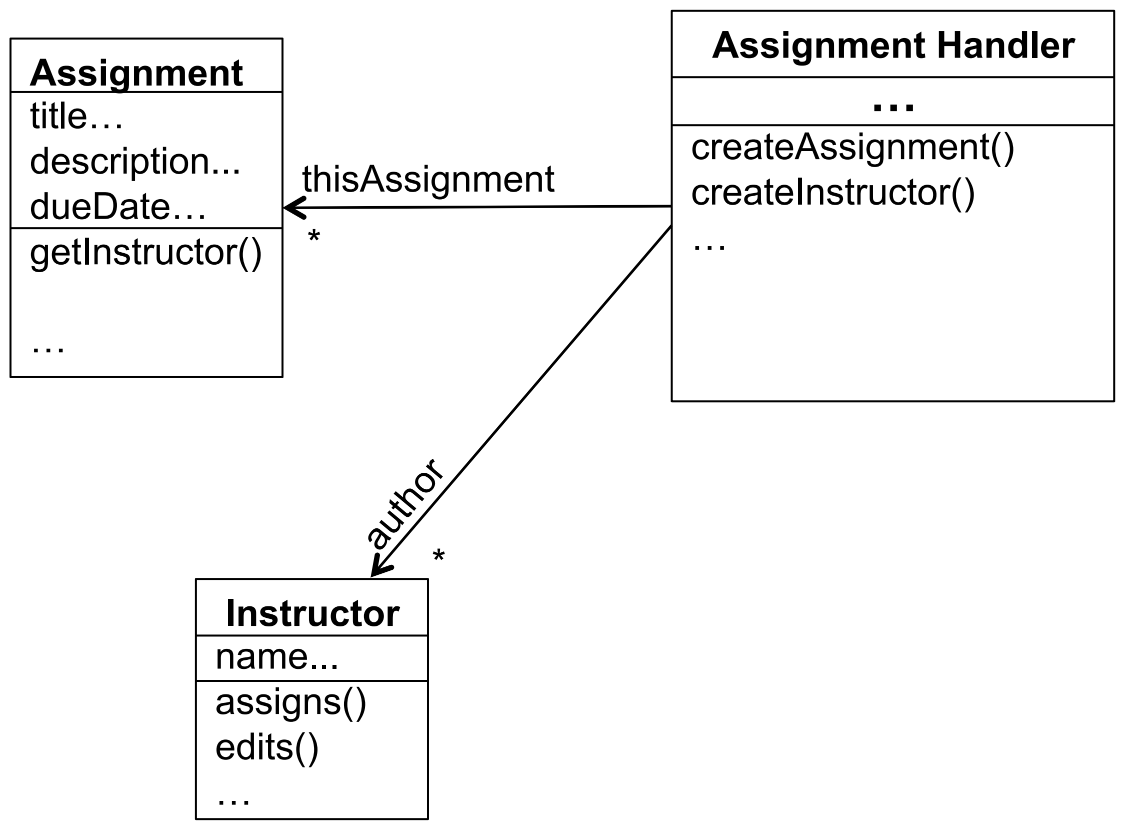
ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Q6

CD Solution for createNewAssignment


creatNewAssignment (title, description, dueDate, authors)➔
by Controller
:Assignment Handler
1: create(title, description, dueDate, authors)➔
by Creator & Expert
:Assignment
1.1: set (t=title, ➔ desc=description, due=dueDate)
2: create (authors) ➔
by Creator
:Instructors [i] List<instructor>
2.1*: name = author ➔

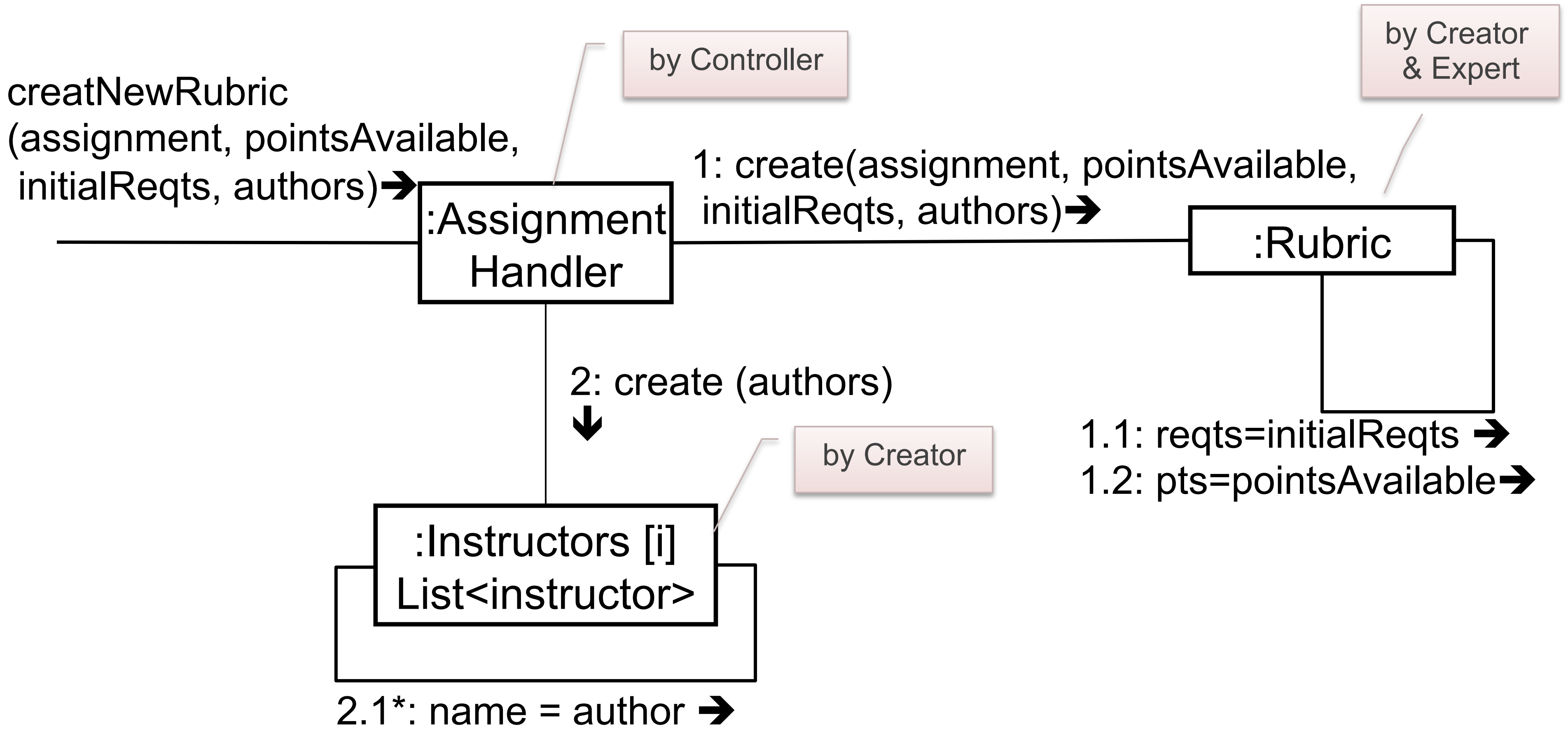
Design Class Diagram



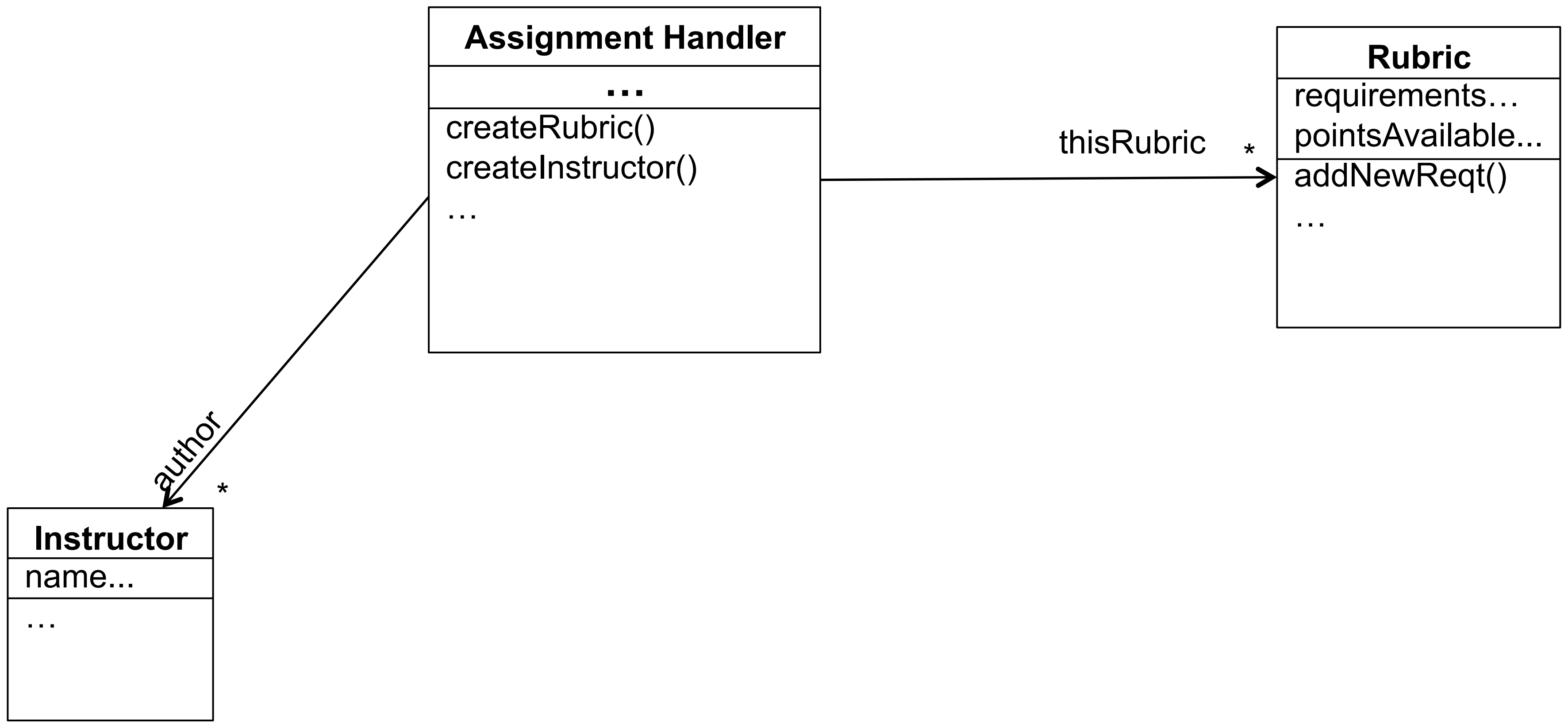
Create New Rubric

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

CD Solution for createNewRubric


creatNewRubric (assignment, pointsAvailable, initialReqts, authors)➔
by Controller
:Assignment Handler
1: create(assignment, pointsAvailable, initialReqts, authors)➔
by Creator & Expert
:Rubric
1.1: reqts=initialReqts ➔
1.2: pts=pointsAvailable➔
2: create (authors) ➔
by Creator
:Instructors [i] List<instructor>
2.1*: name = author ➔

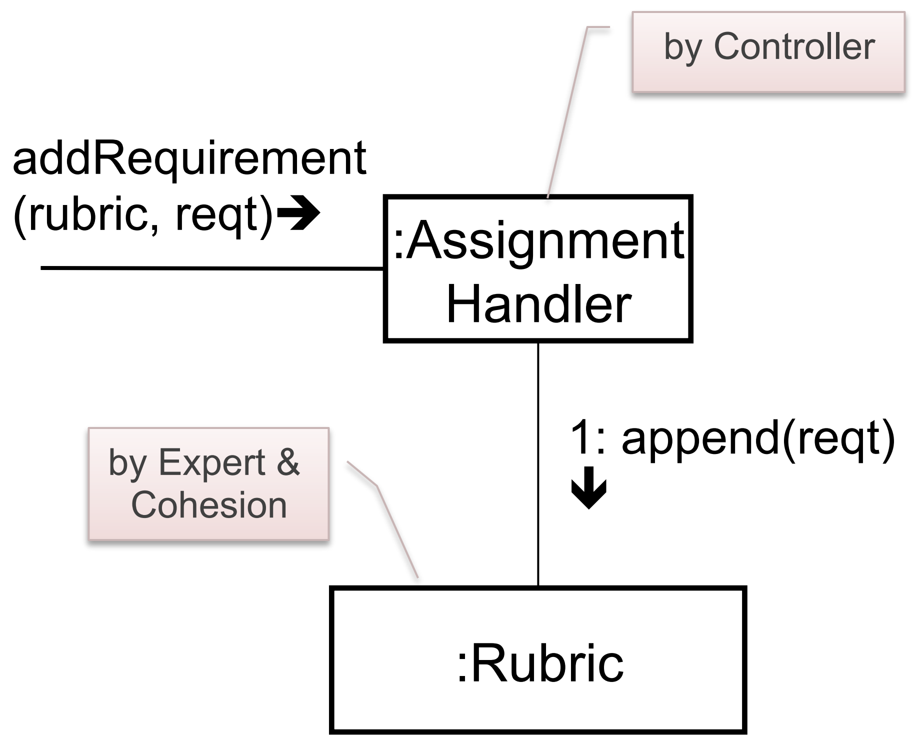
Design Class Diagram

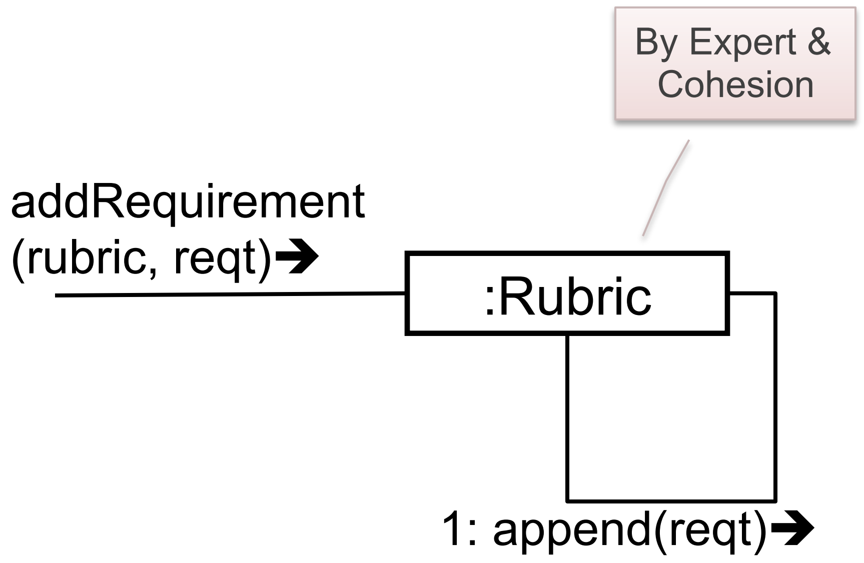


Add Requirement

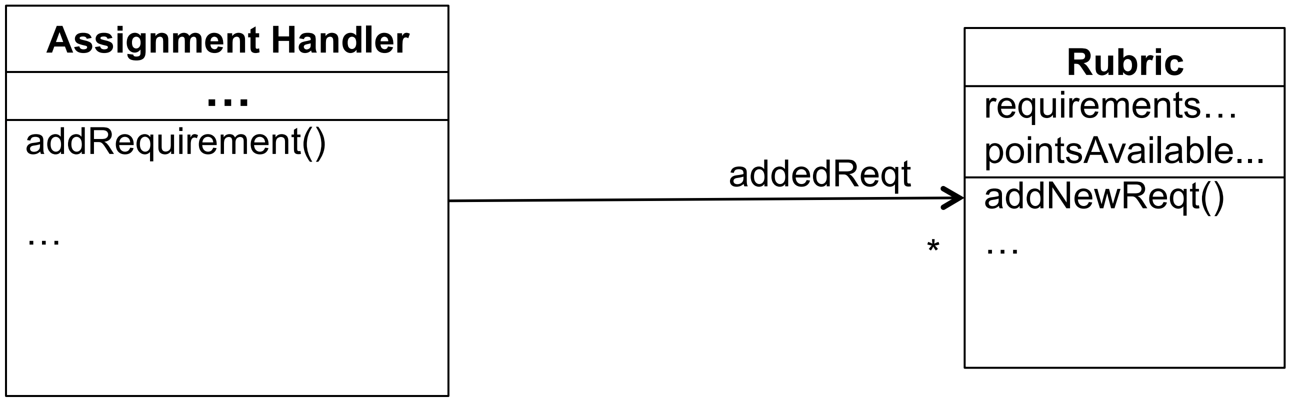
Operation	*addRequirement(rubric, requirement)*
Cross References	Use Case: Create Assignment
Preconditions	*rubric* is an existing *Rubric* in the system
Postconditions	▪ *requirement* was appended to *rubric.requirements*

CD Solution for addRequirement

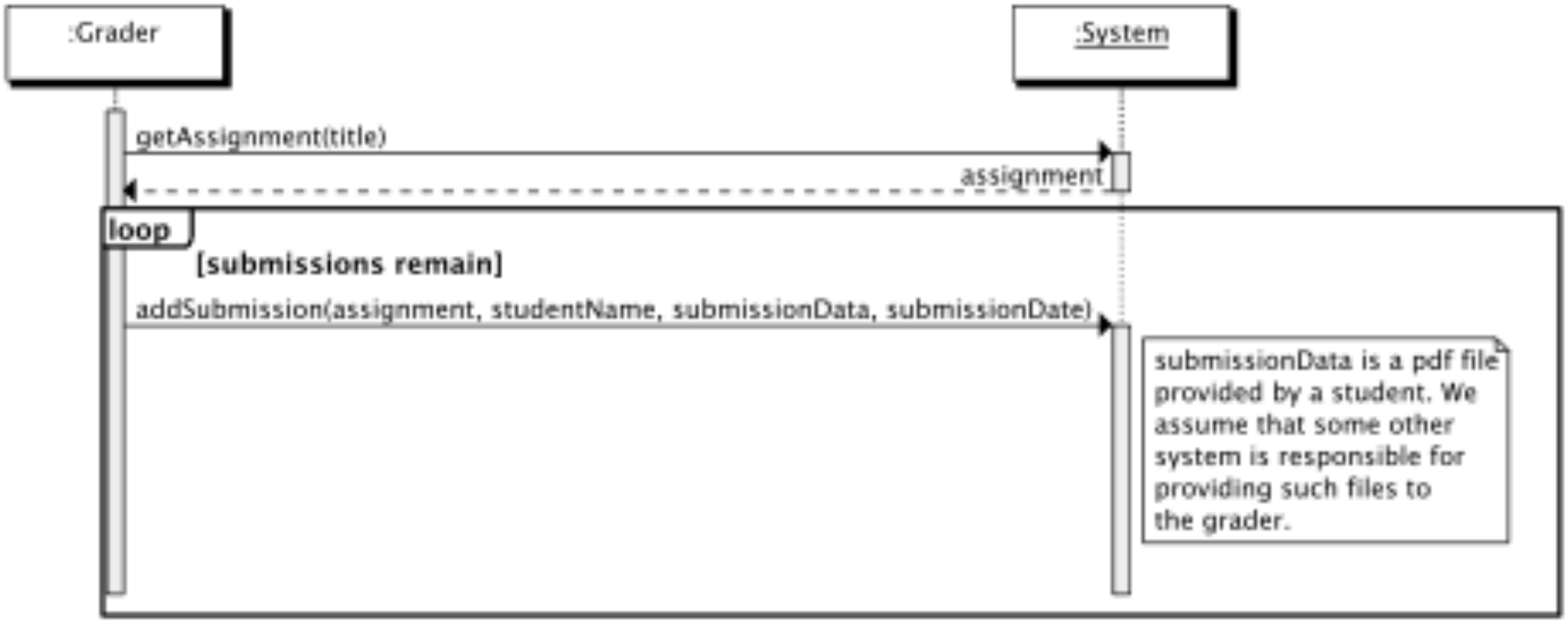




Design Class Diagram

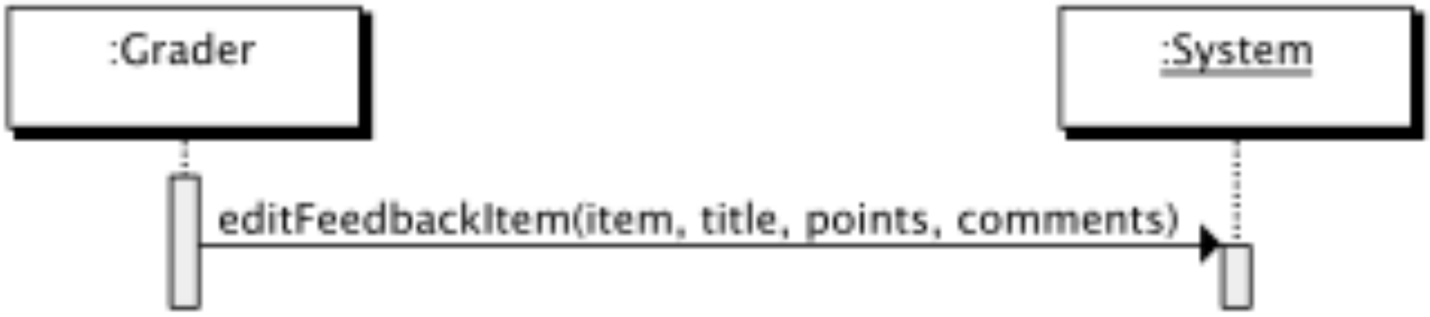


Import Student Submissions Scenario



:Grader
:System
getAssignment(title)
assignment
loop
[submissions remain]
addSubmission(assignment, studentName, submissionData, submissionDate)
submissionData is a pdf file provided by a student. We assume that some other system is responsible for providing such files to the grader.

Edit Feedback Item Scenario



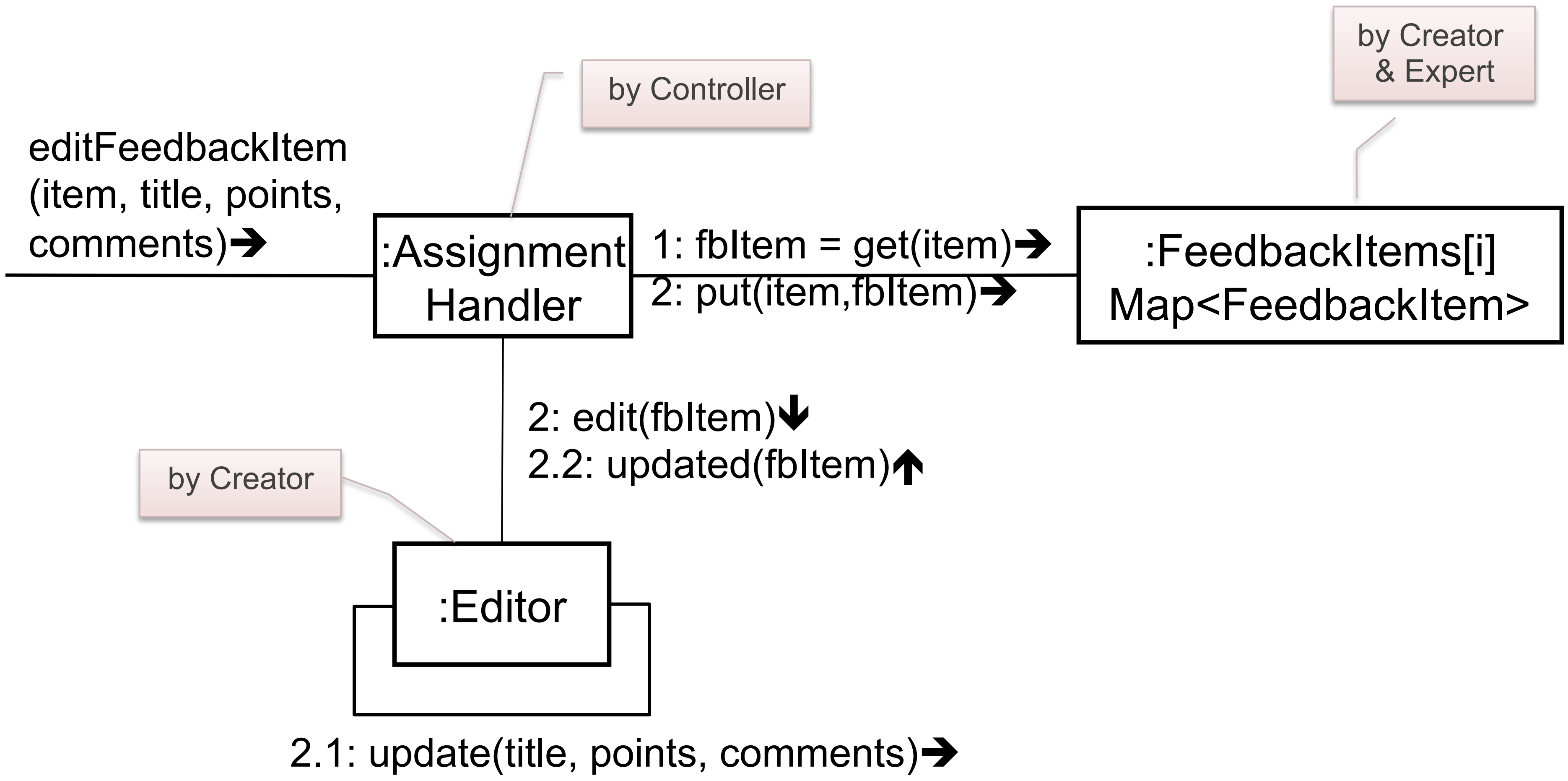
Edit Feedback Item

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

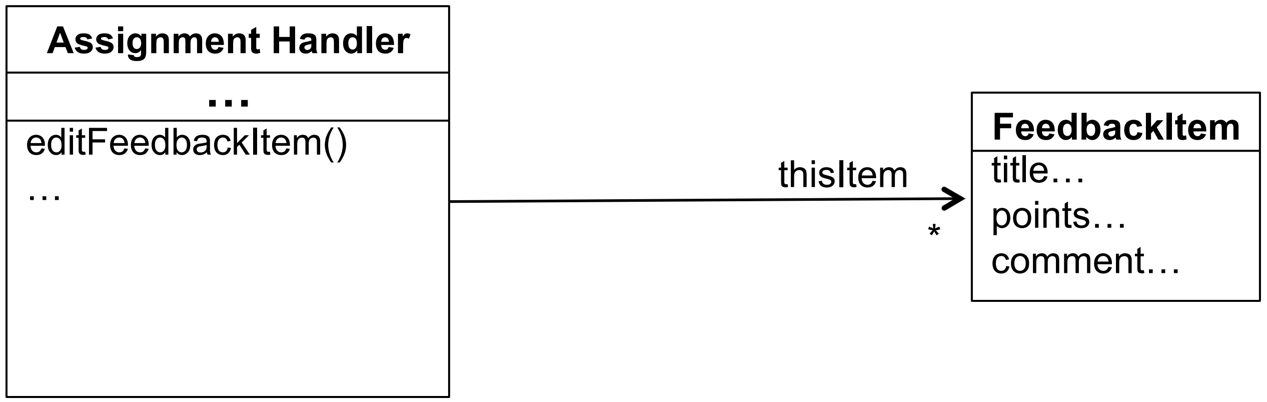
ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Q7

CD Solution for editFeedbackItem



Design Class Diagram



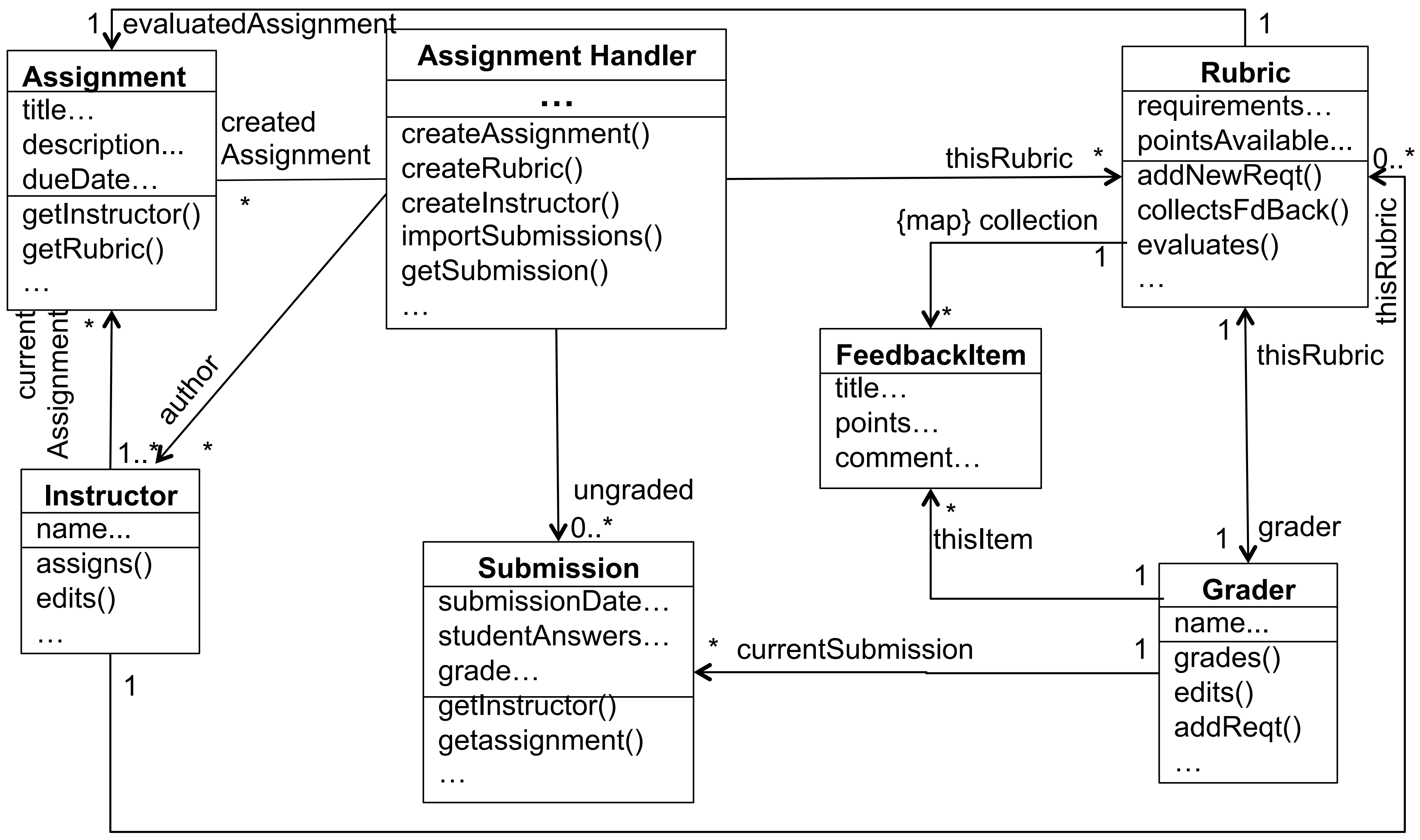
Exercise on Design Examples

- Break up into your project teams
- Given the:
  - Previous DM and SSDs
  - Following OC
- Sketch a communication diagram for the found message, *addSubmission(assignment, studentName, submissionData, submissionDate)*.

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.* ▪ *submission.Date* was set to *submissionDate* ▪ *submission* was associated with *assignment* ▪ a new *Student* instance, *student*, was created ▪ *student.name* was set to *studentName* ▪ *submission* was associated with *student*

Design Class Diagram



Progression From Analysis into Design

- Use Cases drove the development of
  - Domain Model (DM), System Sequence Diagrams (SSD), and 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 directed at controller objects
- Use OC <u>post-conditions</u> to help determine…
  - What should happen in the interaction diagrams
  - What classes belong in the design class diagram

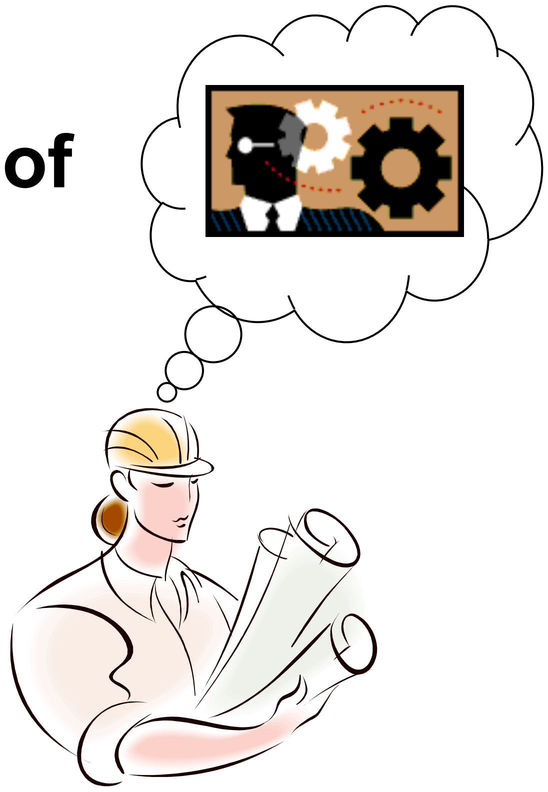
Basic Structure of Thursday's Exam

- 10-15 minutes of breadth (multiple choice and short answer)
- Rest 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"

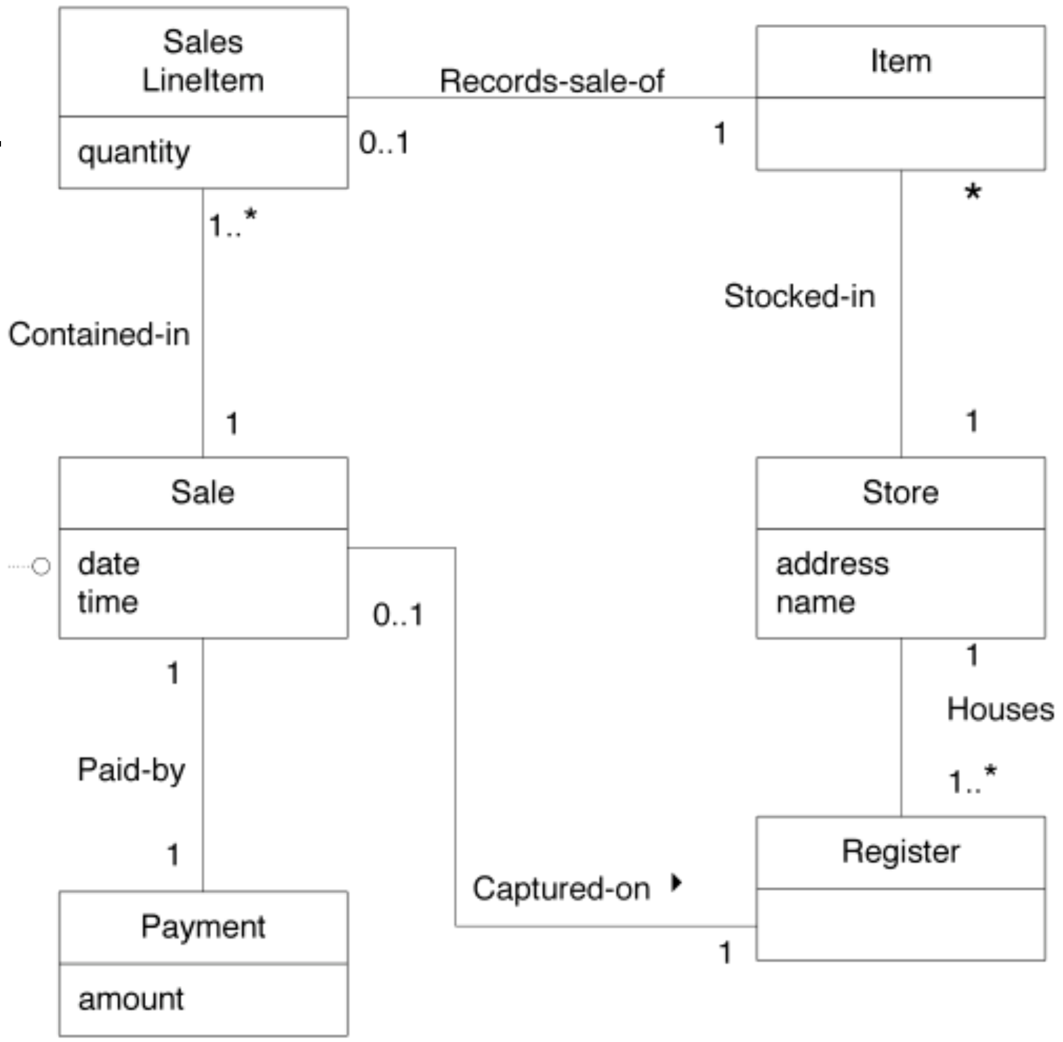


Ways to use Unified Modeling Language (UML)

- 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



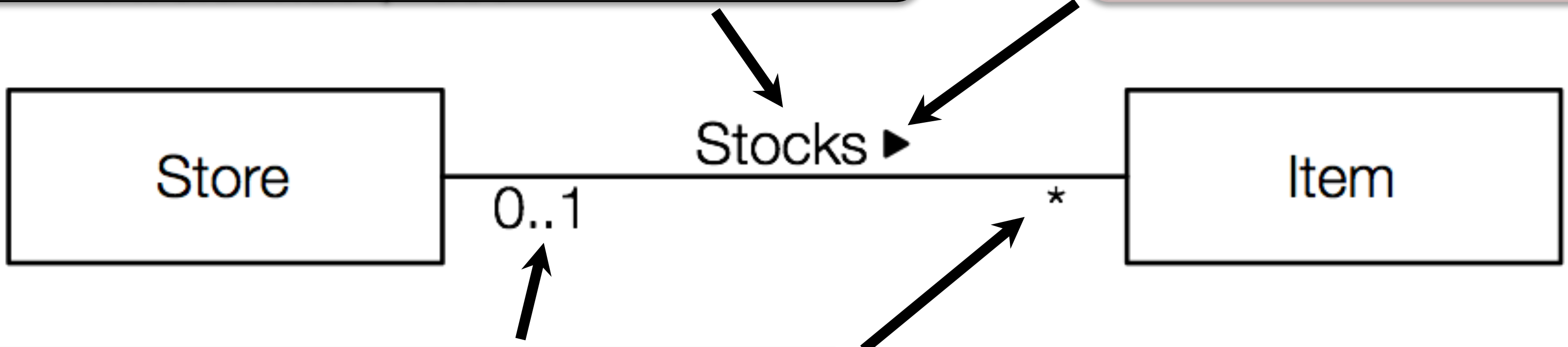
Strategies to Find Conceptual Classes

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

Associations

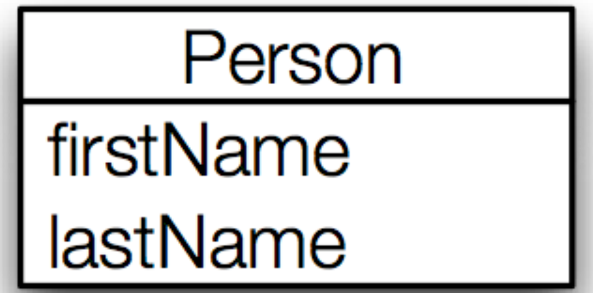
Association name:
- ✓ Use verb phrase
- ✓ Capitalize
- ✓ Typically camel-case or hyphenated
- ✓ Avoid “has”, “use

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



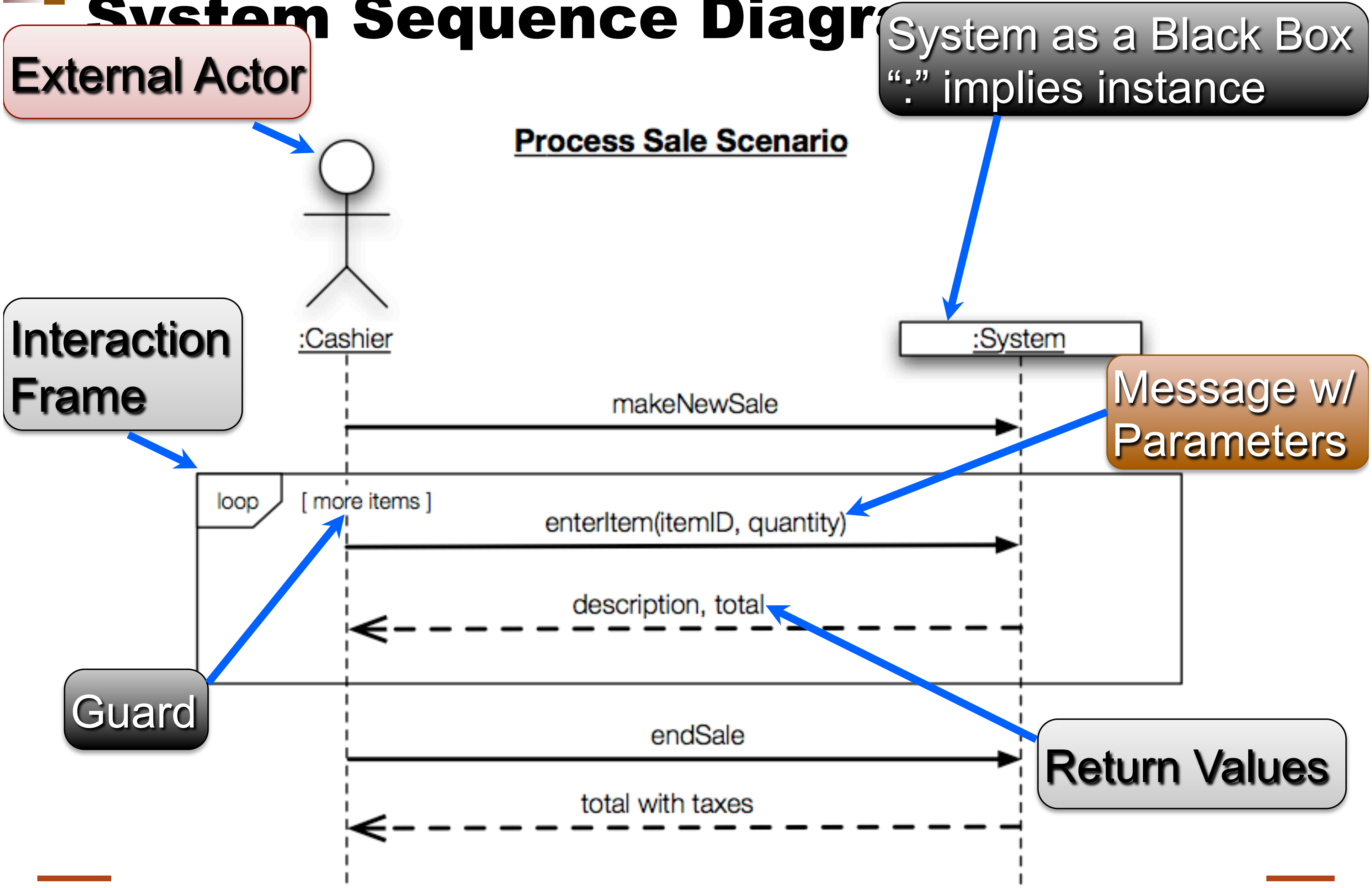
- Include attributes that the requirements suggest need to be remembered
- The usual ‘primitive’ data types
- Common compound data types
- Notation (“[ ]”indicate optional parts):
  - [+|-] [/] *name* [: [*type*] [*multiplicity*]] [= *default*] [{*property*}]

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 Diagr



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.

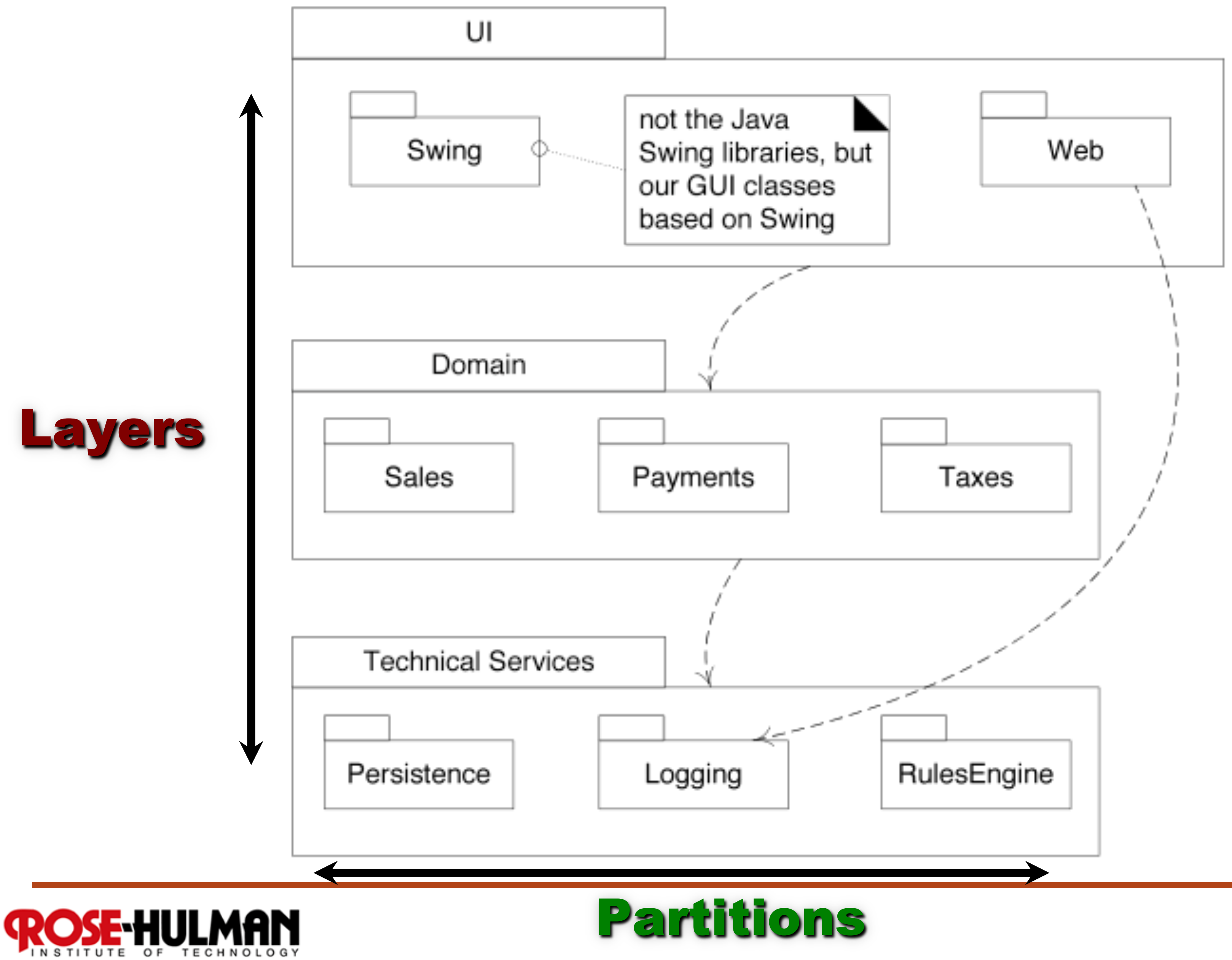
Preconditions: Noteworthy assumptions about the state of the system or objects in the Domain Model before execution of the operation.

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

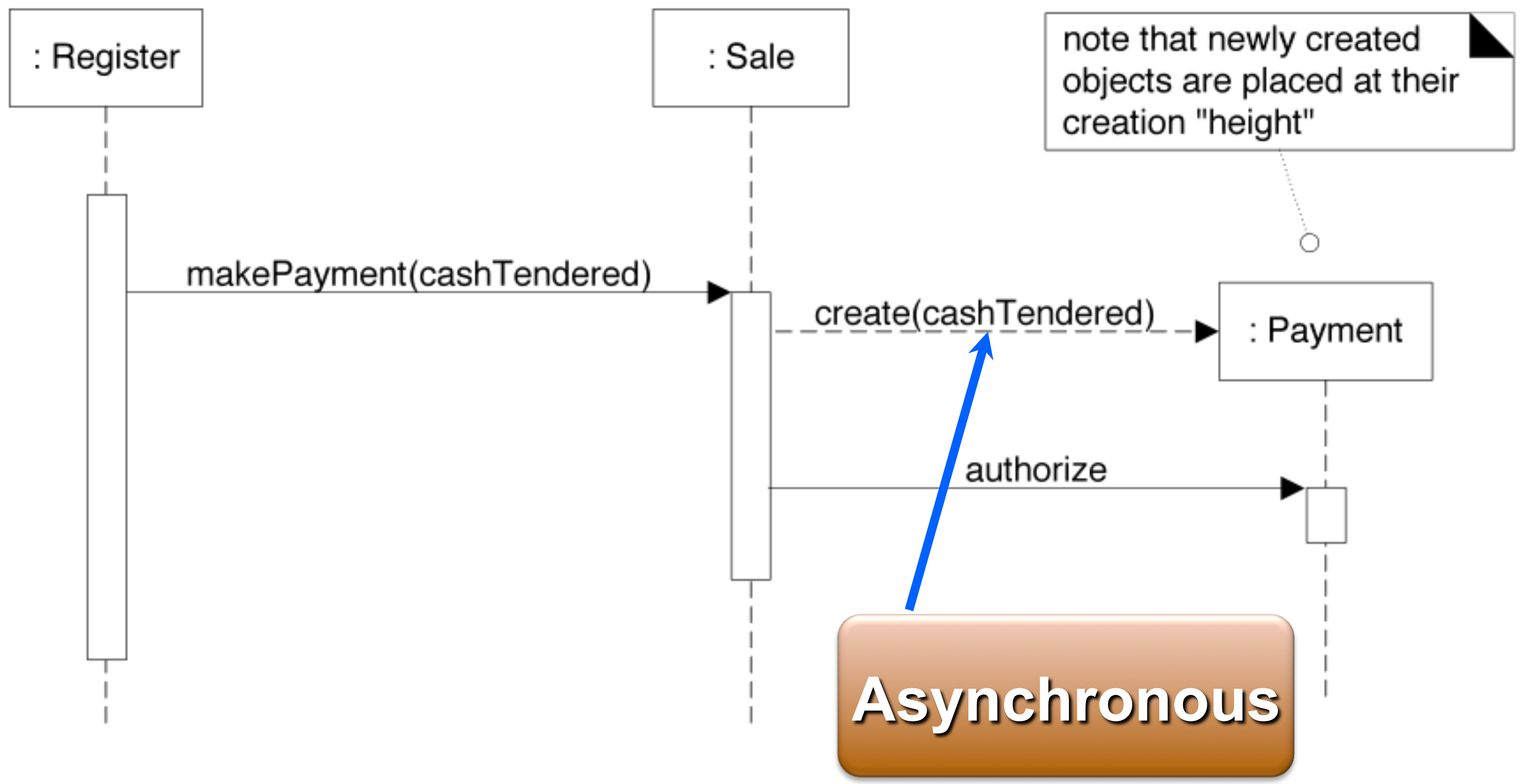
Logical Architecture



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

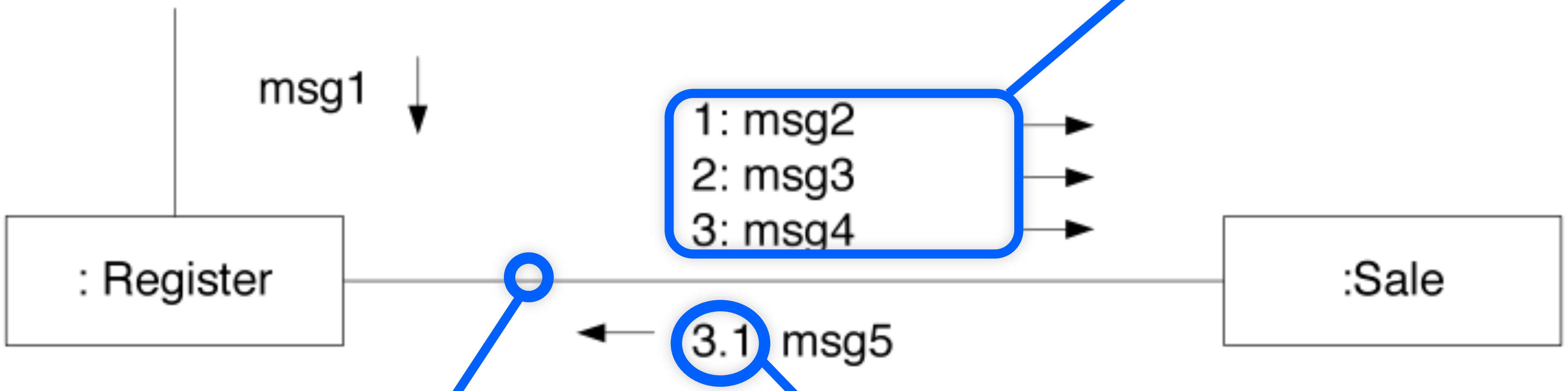

: Register
: Sale
note that newly created objects are placed at their creation "height"
makePayment(cashTendered)
create(cashTendered)
: Payment
authorize
Asynchronous

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”

Communication Diagrams

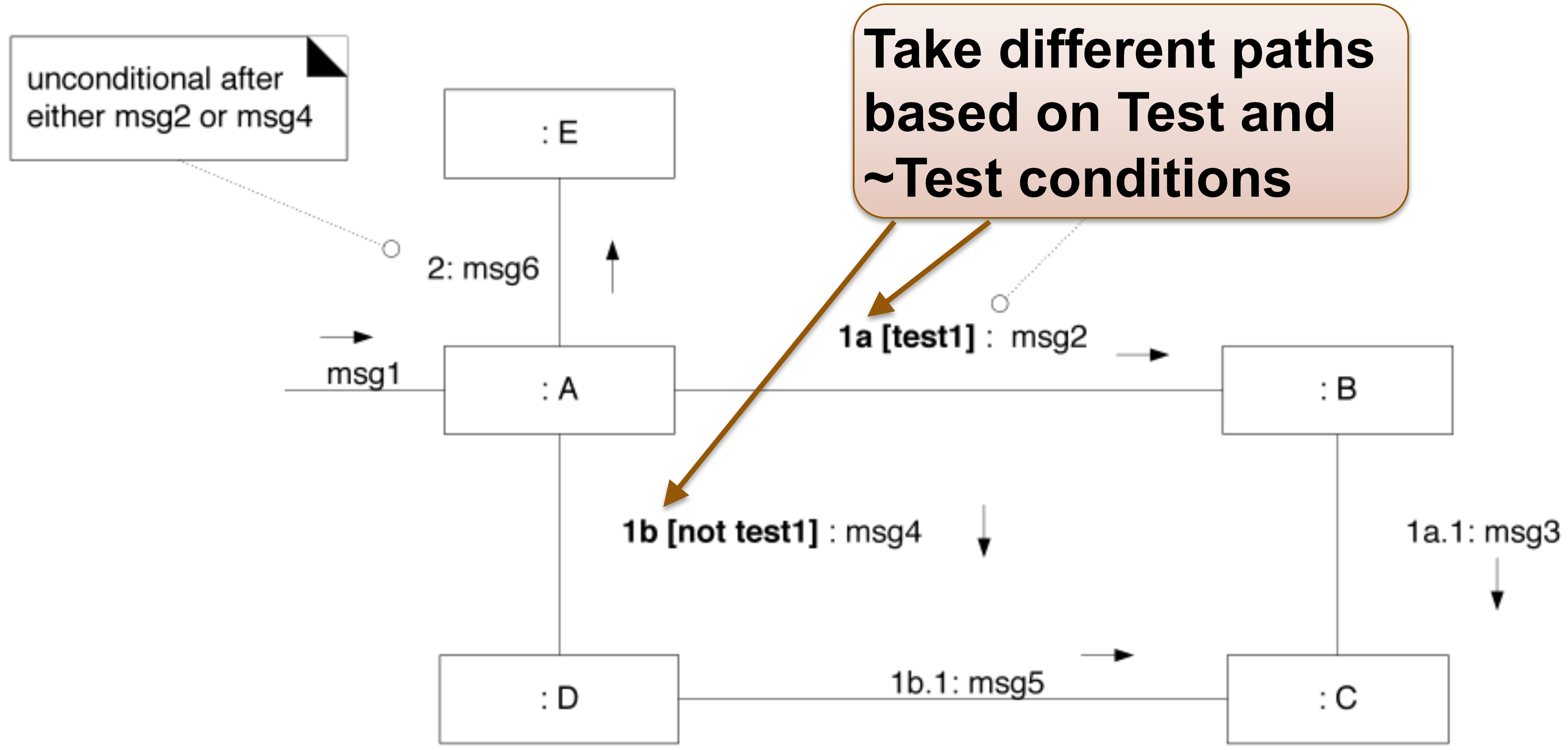
Multiple **messages** traverse links



Single link connects two objects

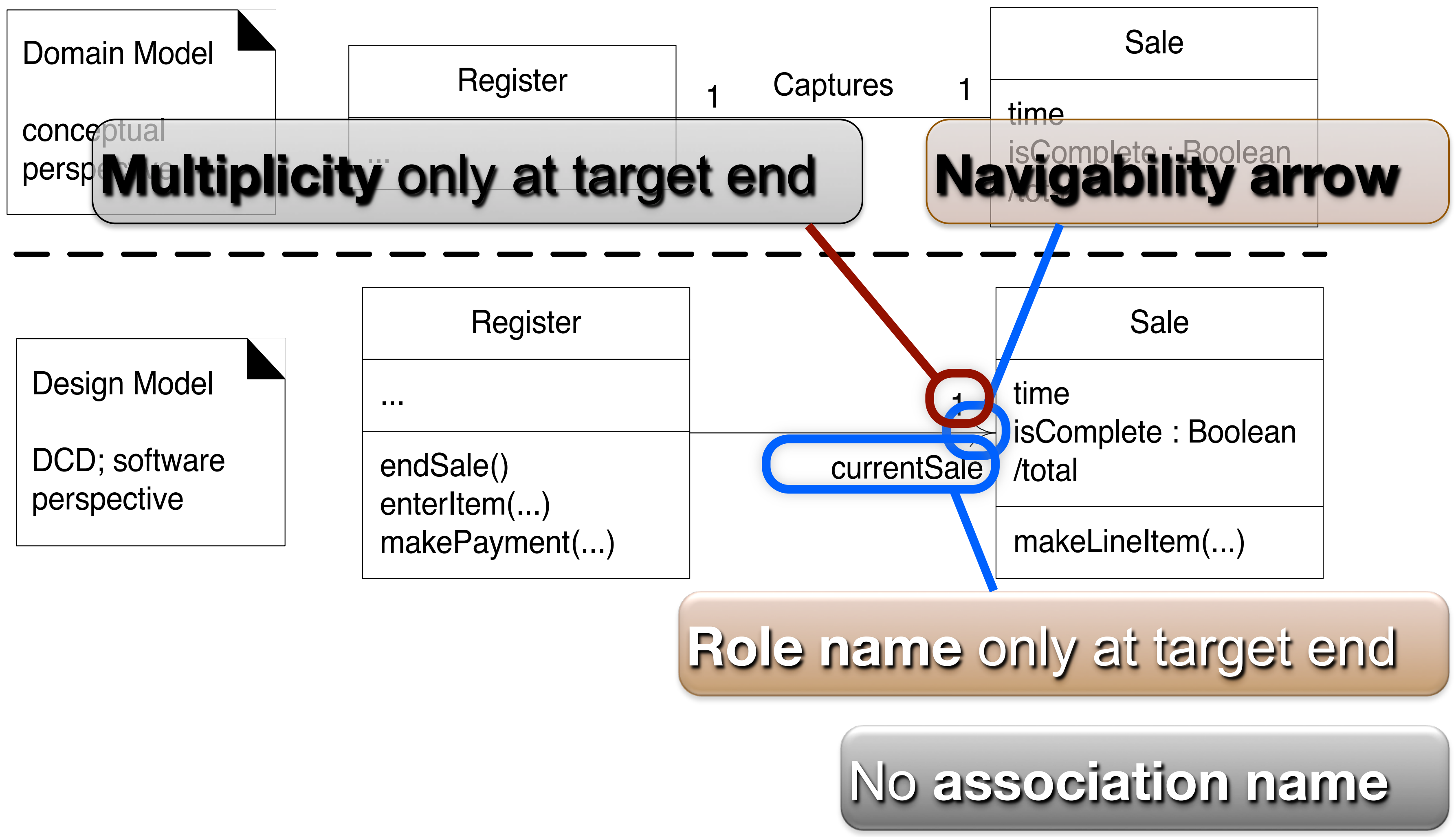
Sequence number gives ordering

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 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

RDD: Knowing & Doing Responsibilities

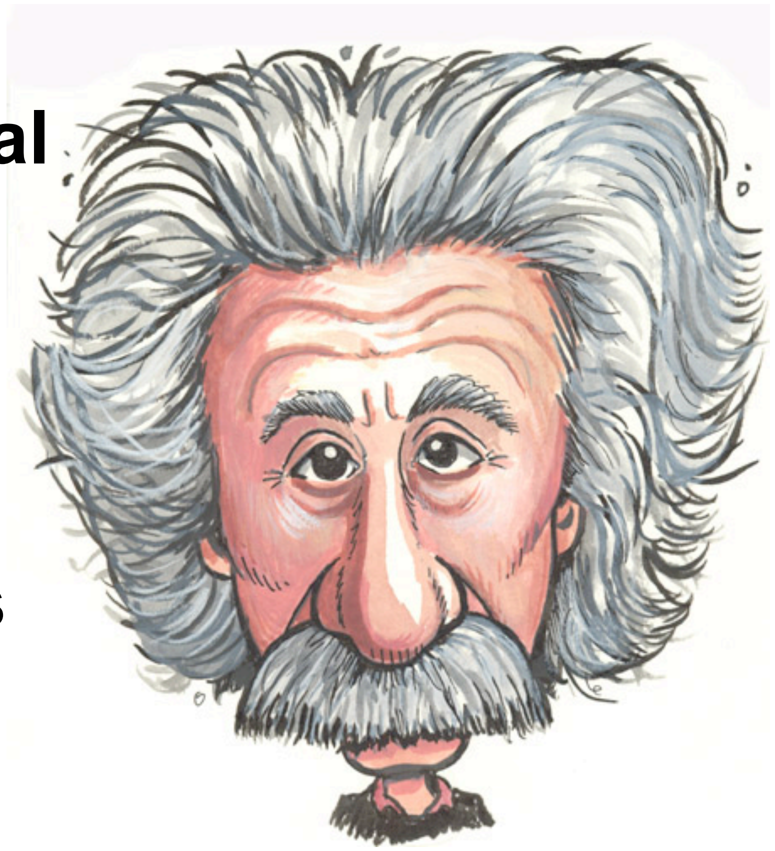
- "Doing" Responsibilities
  - Create another object
  - Perform a calculation
  - Initiate an action in an object
  - Control/coordinate activities of objects
- "Knowing" Responsibilities
  - Knowing it's own encapsulated data
  - Knowing about other objects
  - Knowing things it can derive or calculate

GRASP: 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*

GRASP: Information Expert

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



GRASP: Controller

- Problem: What is the first object beyond the UI layer that 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

GRASP: Low Coupling

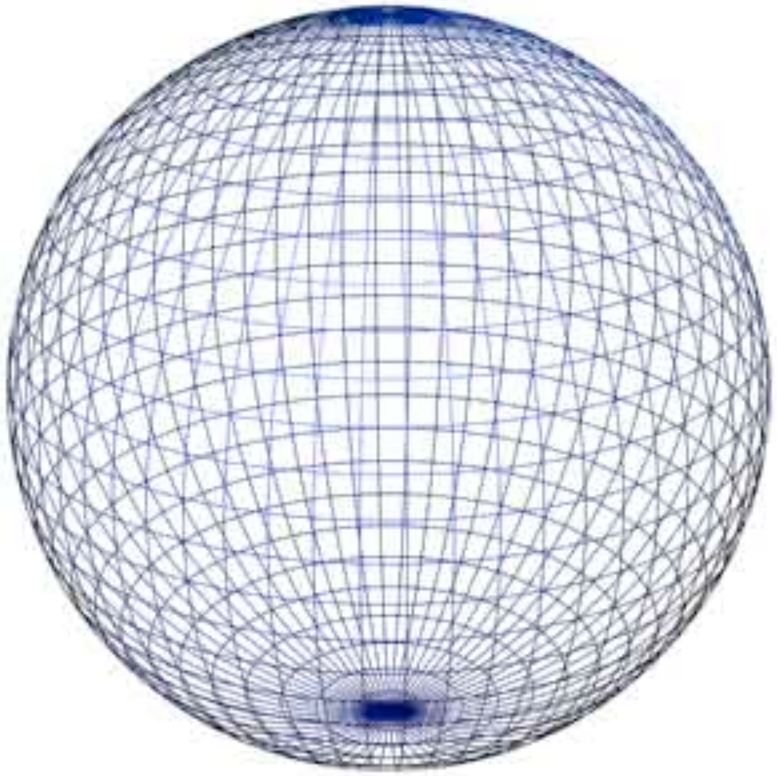
Problem: How do you support low dependency, low change impact, and increased reuse?

Solution: Assign a responsibility so that coupling remains low. Use this principle to evaluate alternatives.

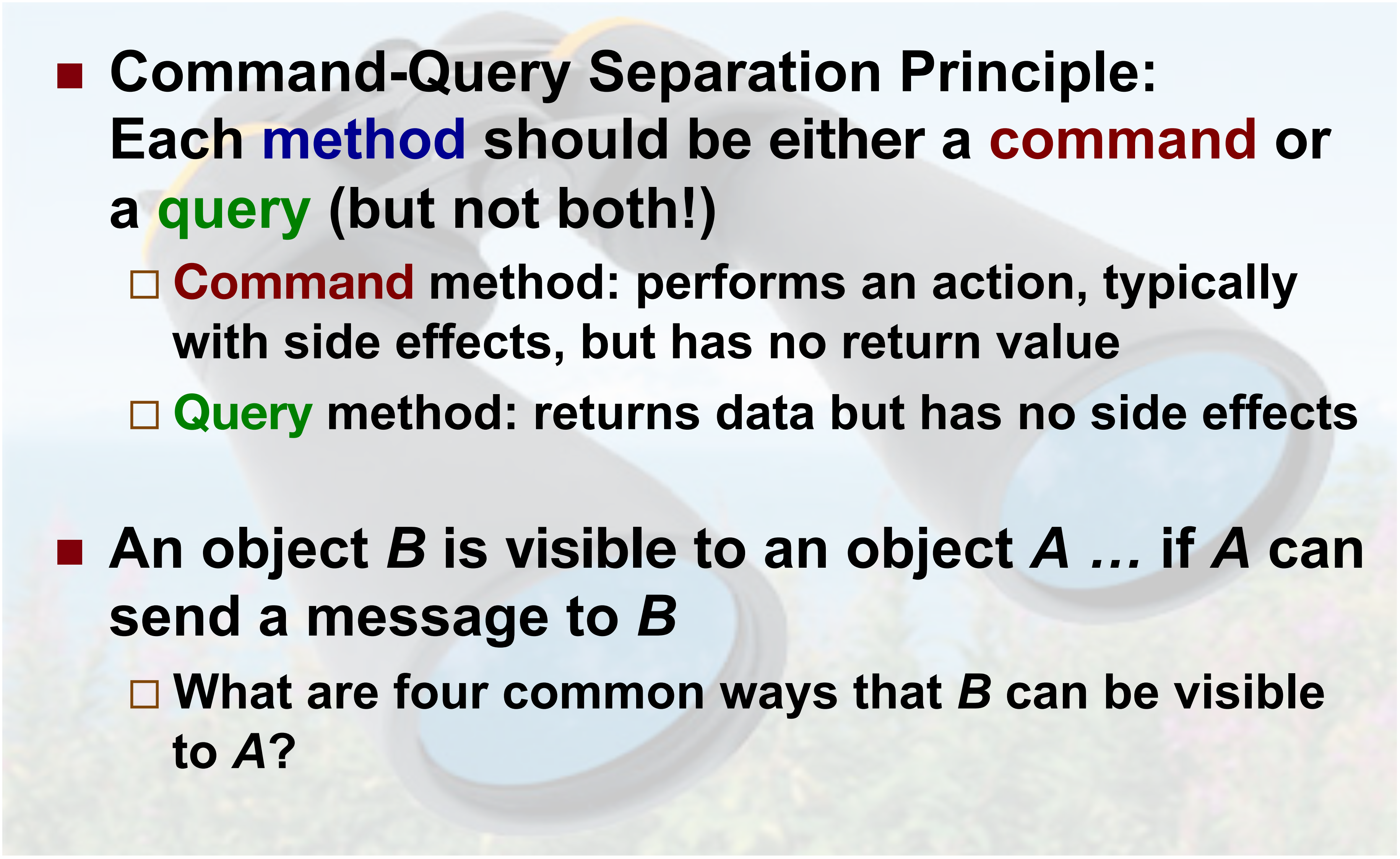
GRASP: High Cohesion

Problem: How do you keep objects focused, understandable, and manageable, and as a side-effect, support low coupling?

Solution: Assign a responsibility so that cohesion remains high. Use this principle to evaluate alternatives.



CQS and Visibility



- Command-Query Separation Principle: Each method should be either a command or a query (but not both!)
  - Command method: performs an action, typically with side effects, but has no return value
  - Query method: returns data but has no side effects
- An object *B* is visible to an object *A* … if *A* can send a message to *B*
  - What are four common ways that *B* can be visible to *A*?

Homework and Milestone Reminders

- Homework 4 – BBVS Design using GRASP and Midcourse Team Evaluation Exercise
  - Due by 11:59pm Tuesday, January 11th, 2011
  - If you want feedback on this before exam, you need to turn it in.
- Study for Examination on Thursday
- Read Chapter 20 on Design to Code for Monday
- Milestone 4 – Junior Project Design with GRASP
  - Due by 11:59pm on Friday, January 28th, 2011