CSSE 374: Object-Oriented Design Exercise

Shawn Bohner

Office: Moench Room F212

Phone: (812) 877-8685

Email: bohner@rose-hulman.edu

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Learning Outcomes: Patterns, Tradeoffs

Identify criteria for the design of a software system and select patterns, create frameworks, and partition software to satisfy the inherent trade-offs.

- Outline Command-Query Separation Principles
- Examine Object Visibility
- Apply OOD to an Extended Example

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

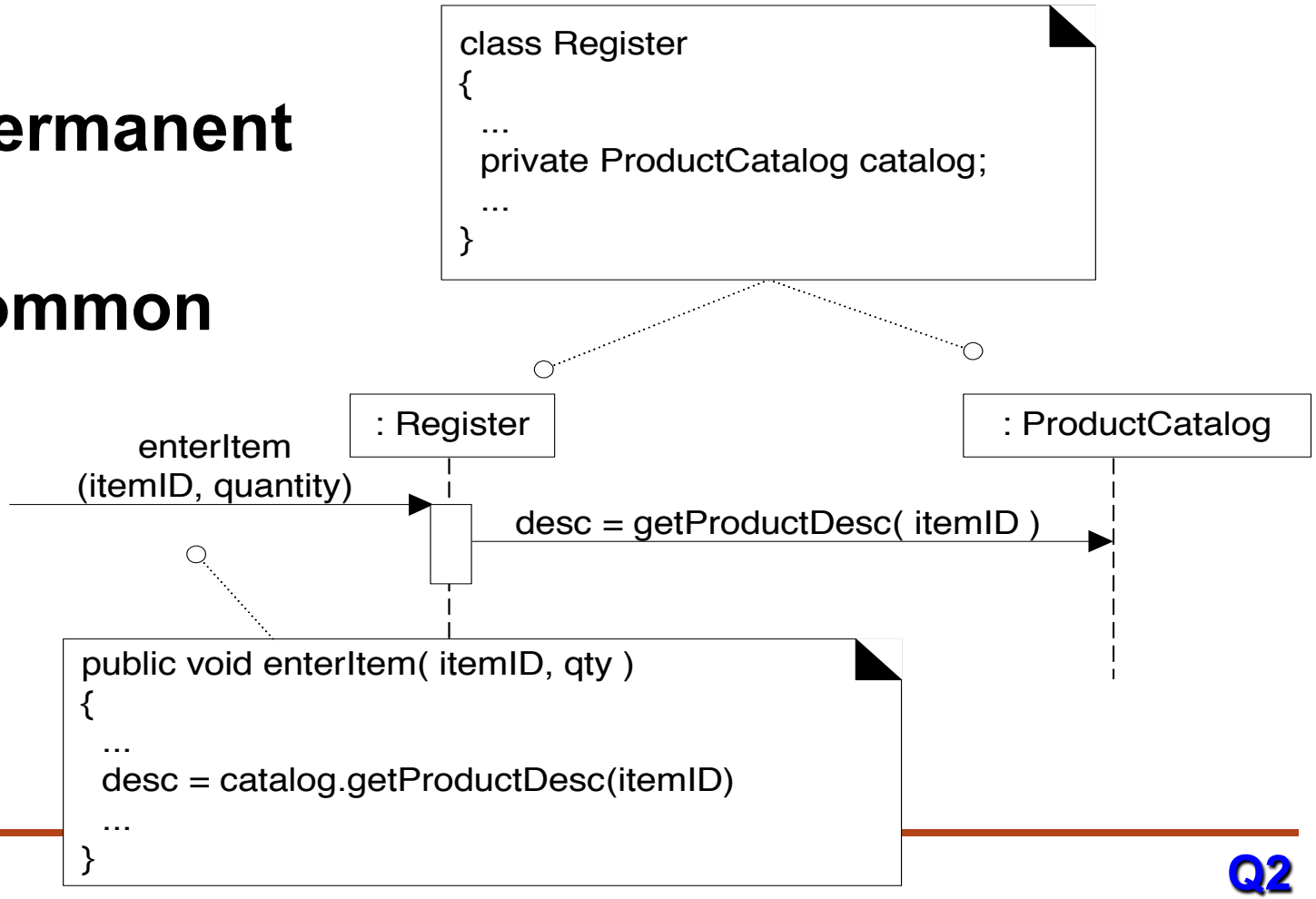
Why?

Visibility

- An object *B* is visible to an object *A* … if *A* can send a message to *B*
- Related to, but not the same as:
  - Scope
  - Access restrictions (*public*, *private*, etc.)
- What are four common ways that *B* can be visible to *A*?

Attribute Visibility

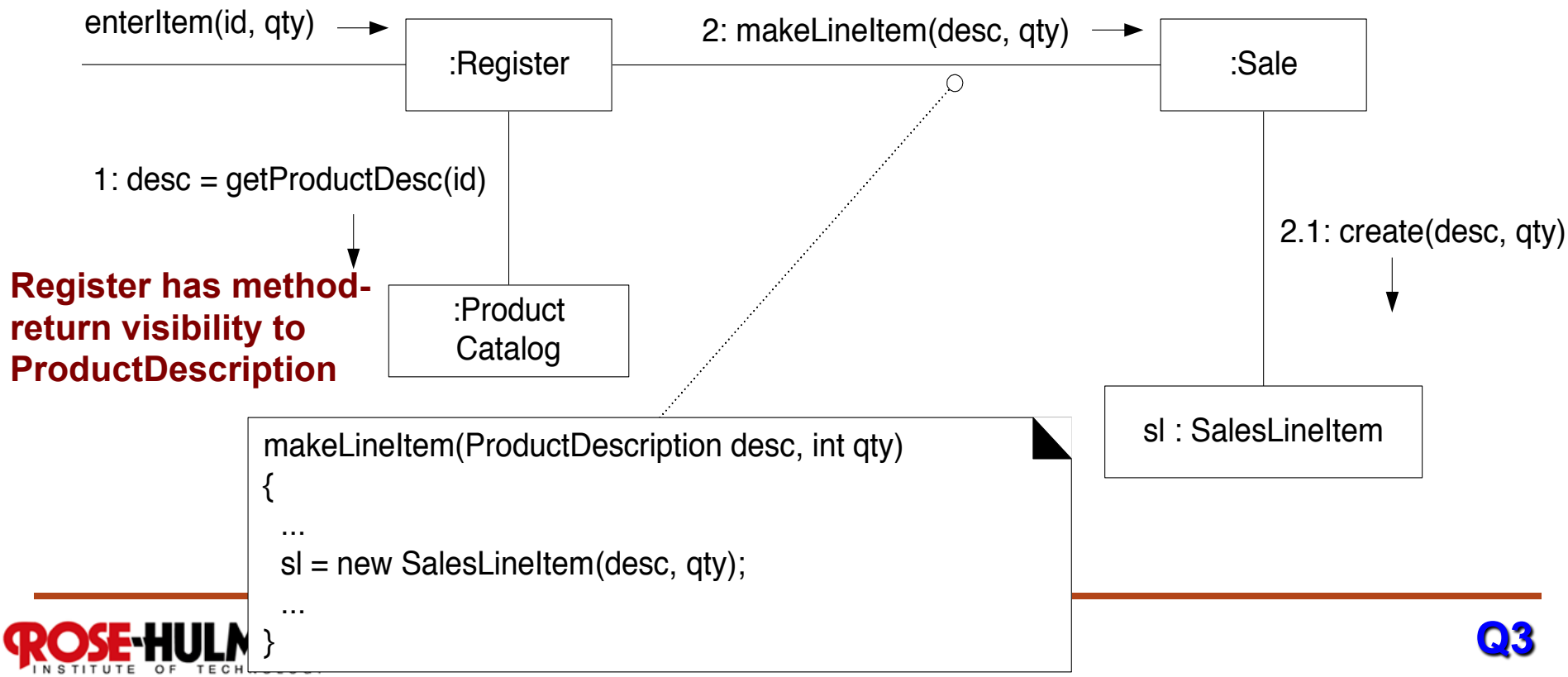
- Object *A* has attribute visibility to object *B* if…
  A has an attribute that stores *B*
- Quite permanent
- Most common



Parameter Visibility

Object *A* has parameter visibility to object *B* if … *B* is passed in as an argument to a method of *A*

- Not permanent, disappears when method ends
- Second most common
- Methods often convert parameter visibility to attribute visibility



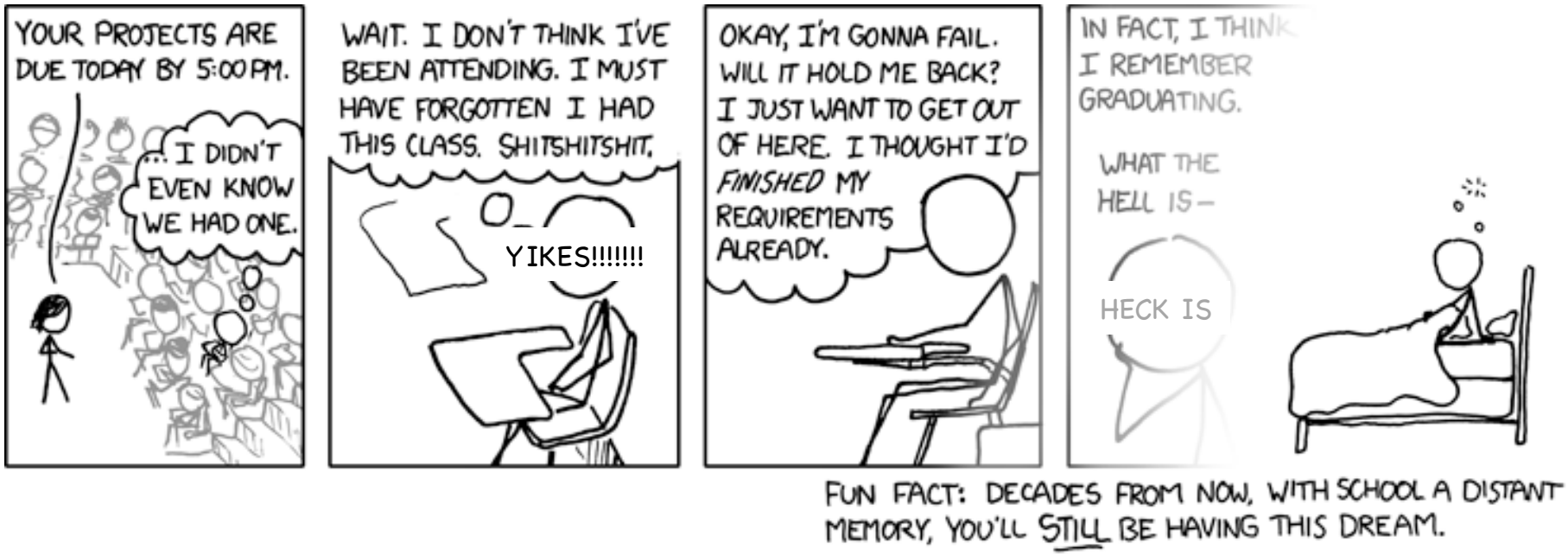
Local Visibility

- Object *A* has local visibility to object *B* if … *B* is referenced by a local variable in a method of *A*
- Not permanent, disappears when leaving variable's scope
- Third most common
- Methods often convert local visibility to attribute visibility

Global Visibility

- Object *A* has global visibility to object *B* if … *B* is stored in a global variable accessible from *A*
- Very permanent
- Least common (but highest coupling risk)

Students



The same goes for the one where you're falling out of the helicopter into the ocean. You guys all have that dream, right? It's not just me…

Extended 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.

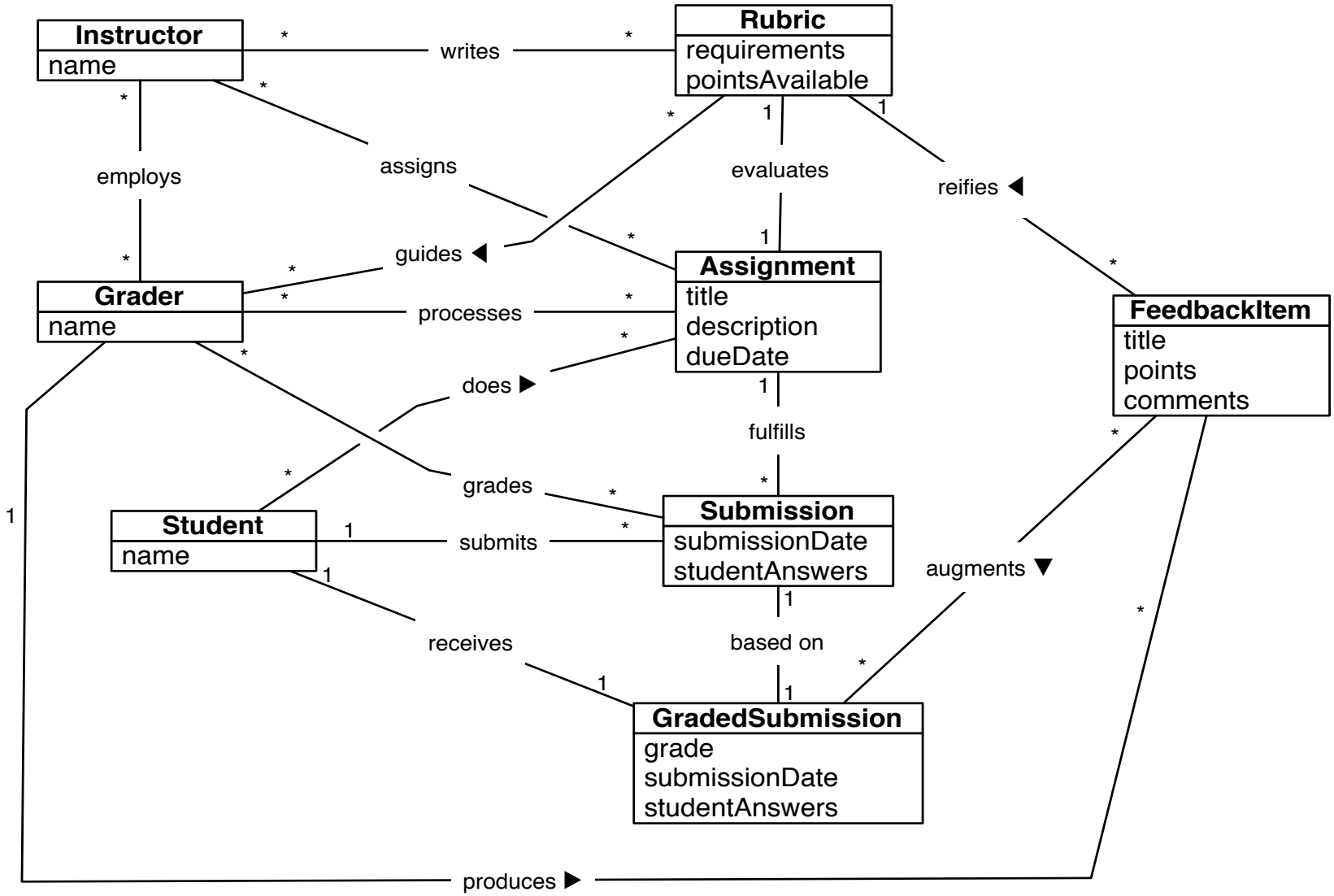
The system will **take a collection of student solutions to an assignment as PDF files** or some other convenient, open standard. It will allow the **grader to "write" feedback on student submissions**. It will **keep track of the grader's place** in each assignment so that he or she can grade every 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.

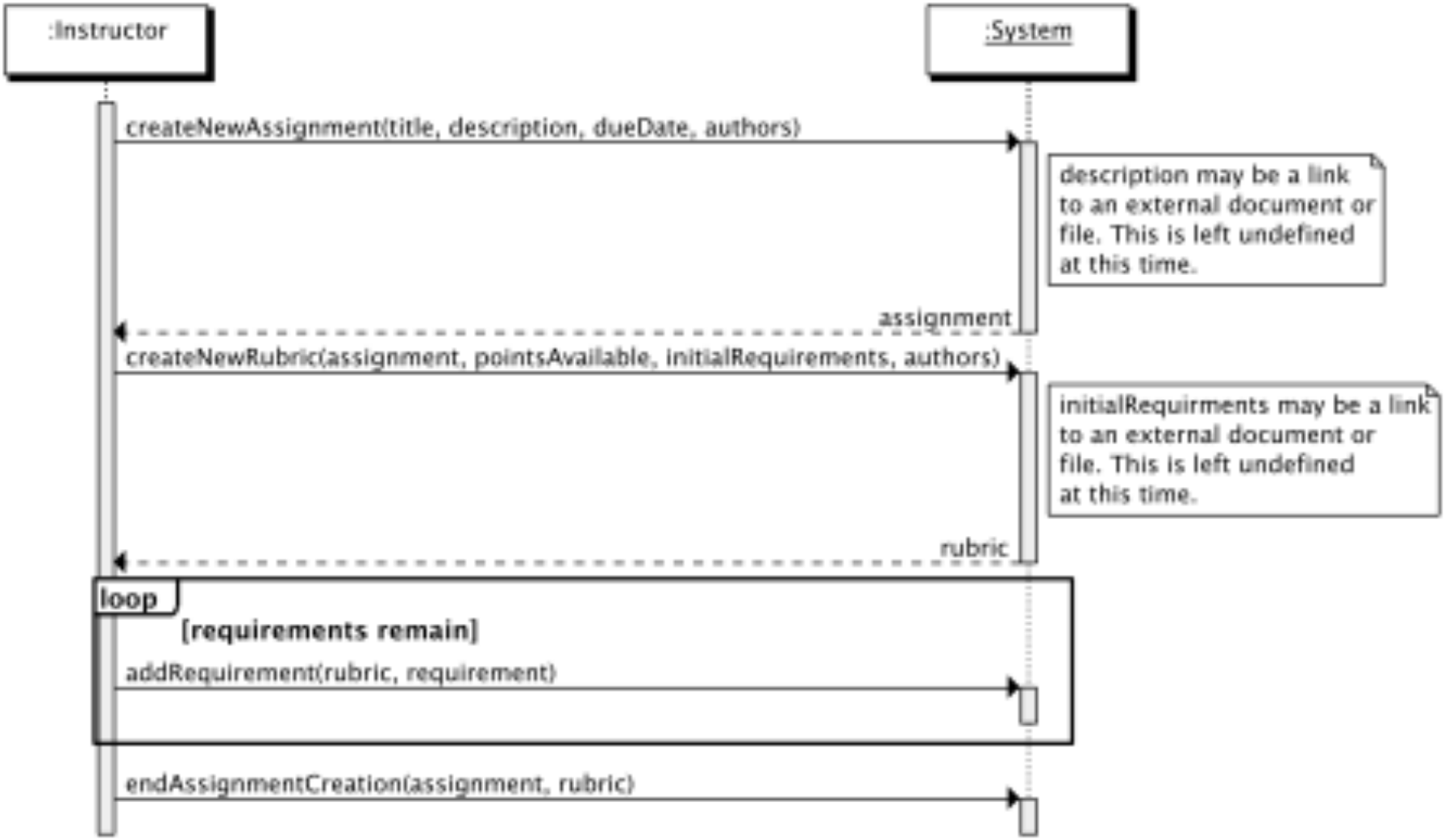
A Sampling of Use Cases

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

Domain Model for Grading System



Create Assignment Scenario



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*

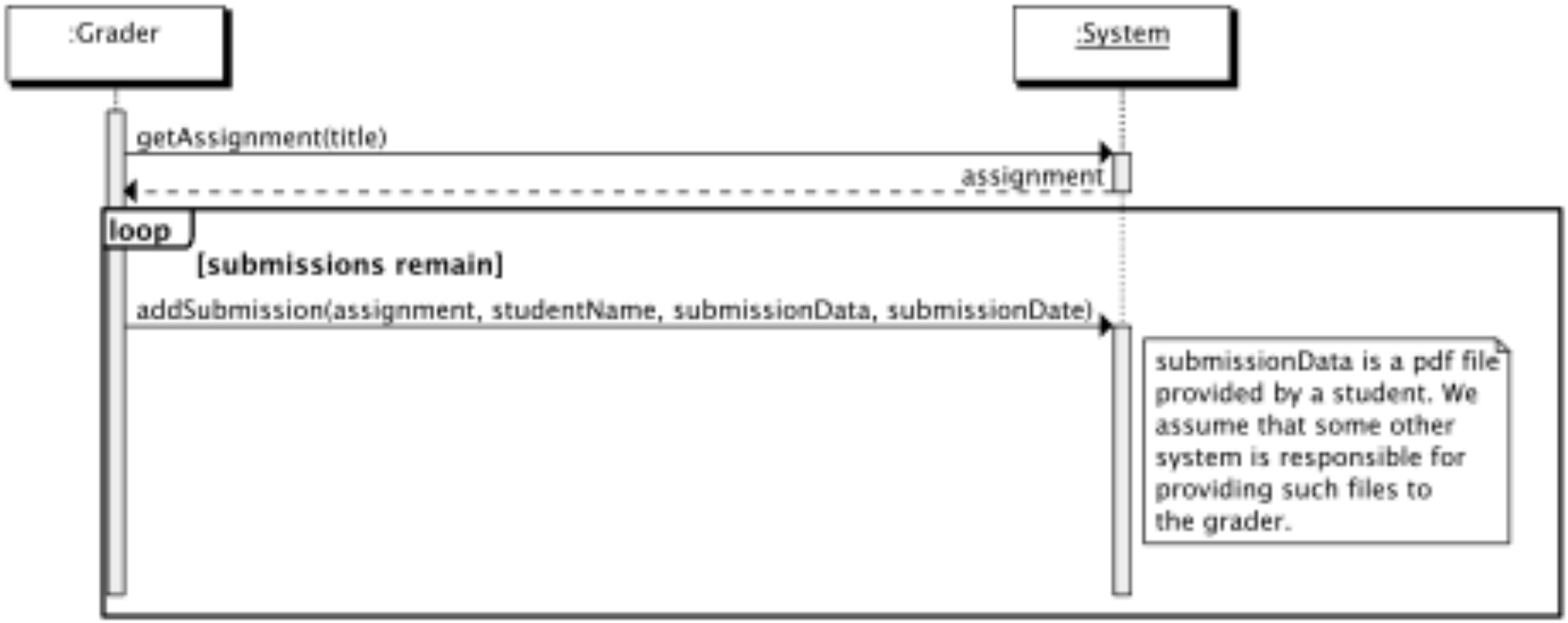
Create New Rubric

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

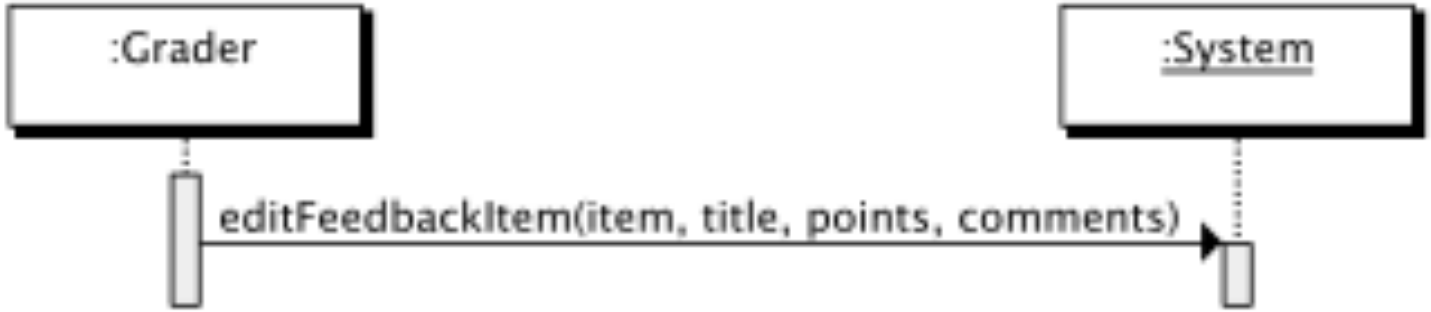
Add Requirement

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***

Import Student Submissions Scenario



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

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*

Homework and Milestone Reminders

- Read Chapter 20 on Design to Code
- 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.

Recall GRASP: Creator

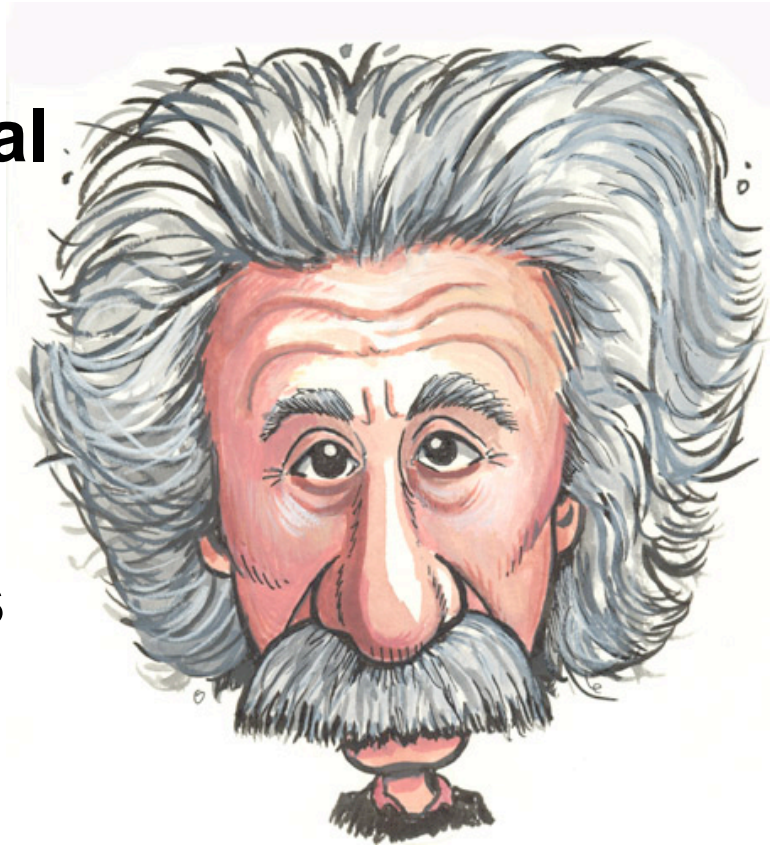
- **Problem**: Who should be responsible for creating a new instance of some class?

- **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*

Recall GRASP: Information Expert

- **Problem: What is a general principle of assigning responsibilities?**

- **Solution: Assign a responsibility to the class that has the necessary information**



Recall 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**

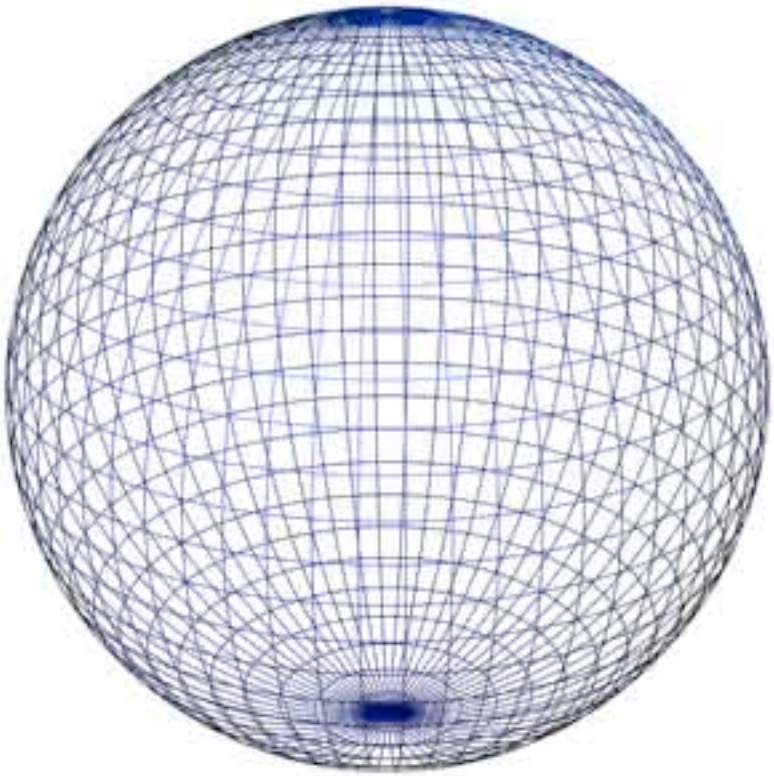
Recall GRASP: Low Coupling

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.

Recall 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.