CSSE 374: GRASP'ing at the First Five ~~Patterns~~ Principles

Shawn Bohner

Office: Moench Room F212

Phone: (812) 877-8685

Email: bohner@rose-hulman.edu

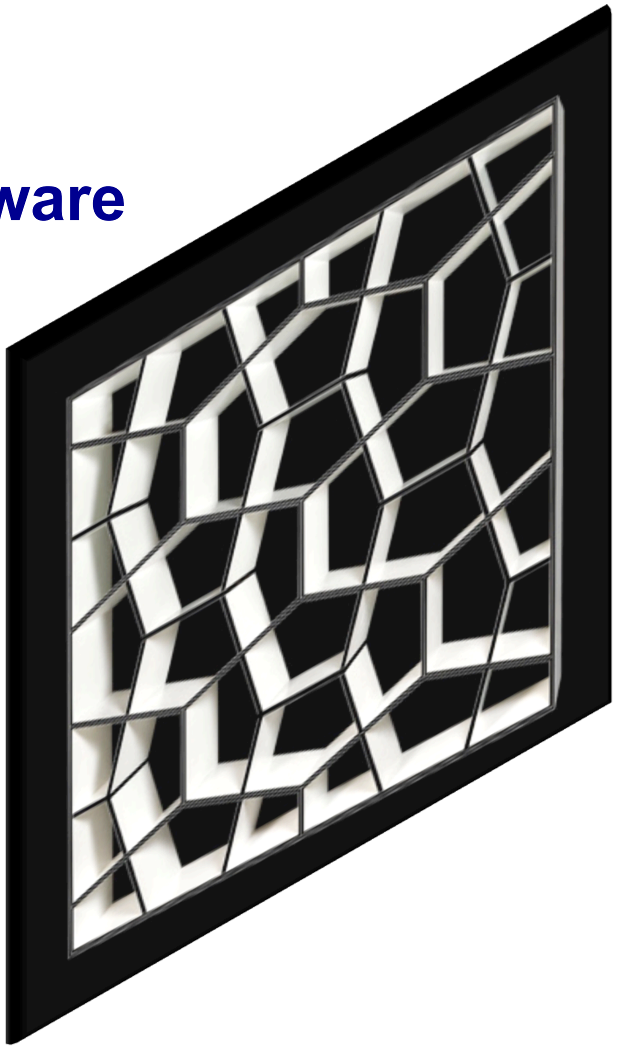
ROSE-HULMAN INSTITUTE OF TECHNOLOGY

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.

Examine GRASP Patterns:

- Creator
- Information Expert
- Controller
- Low Coupling
- High Cohesion



Recall GRASP: Creator

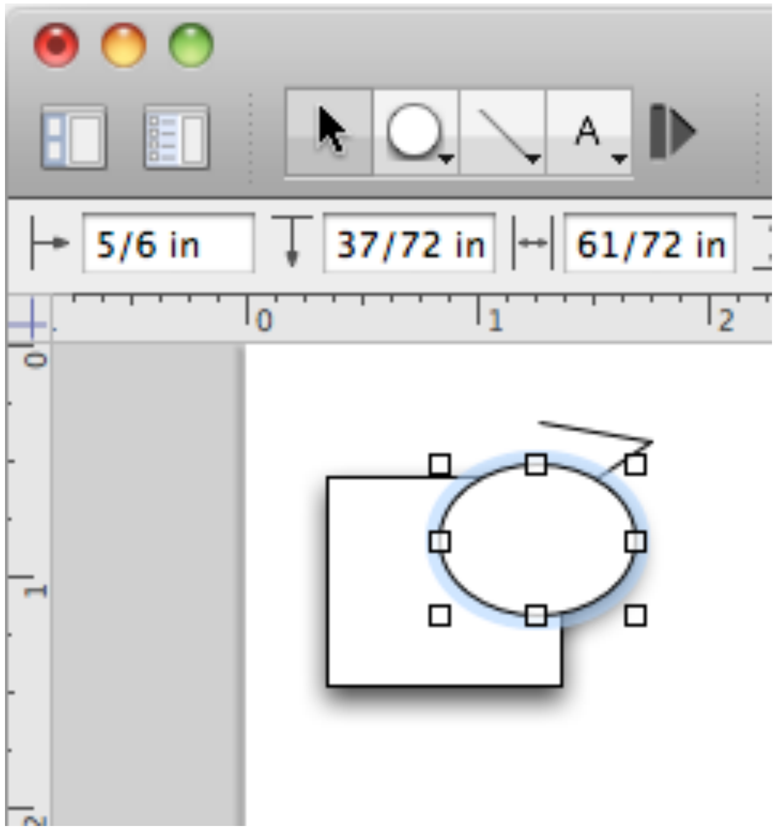
- **Problem: Who should be responsible for creating a new instance of some class?**
- **Solution: Make *B* responsible for creating *A* if…**
  - ***B* contains or is a composition of *A***
  - ***B* records *A***
  - ***B* closely uses *A***
  - ***B* has the data to initialize *A***

Most important

The more matches the better.

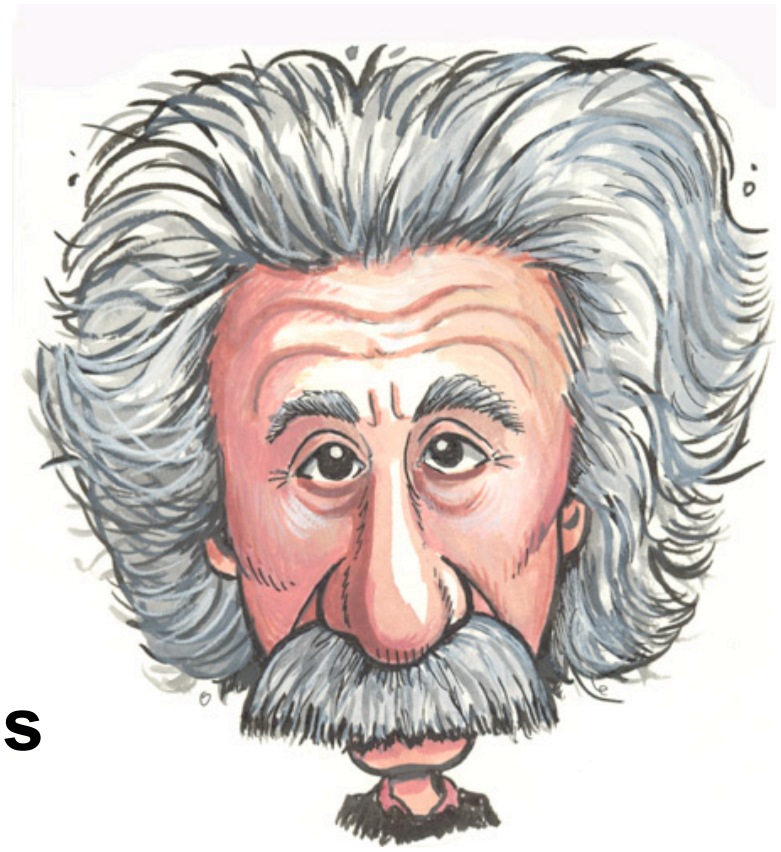
Creator Contraindications

- Complex creation scenarios
  - Recycling instances
  - Conditional creation



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



Information Expert Contraindications

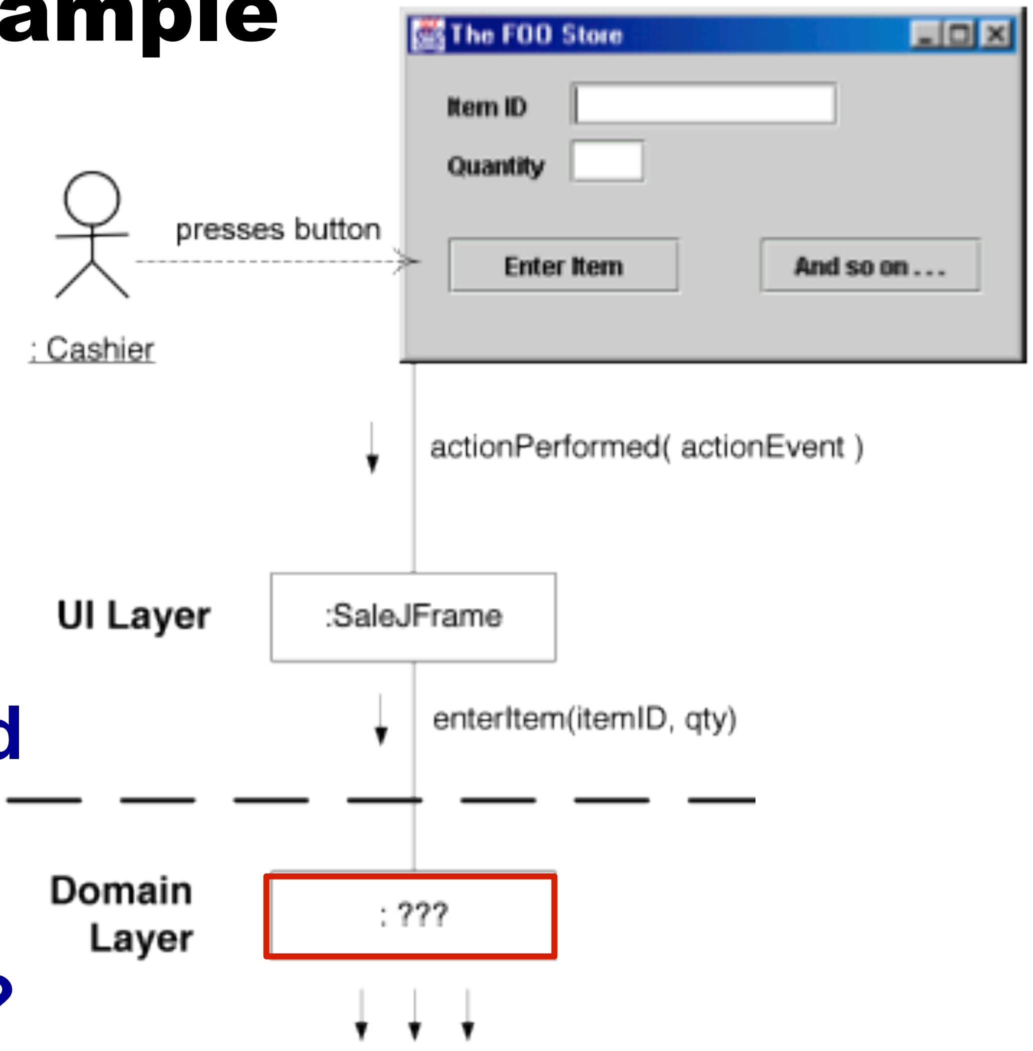
- Sometimes Information Expert will suggest a solution that leads to coupling or cohesion problems
- Consider: Who should be responsible for saving a Sale in a database?

GRASP: Controller

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

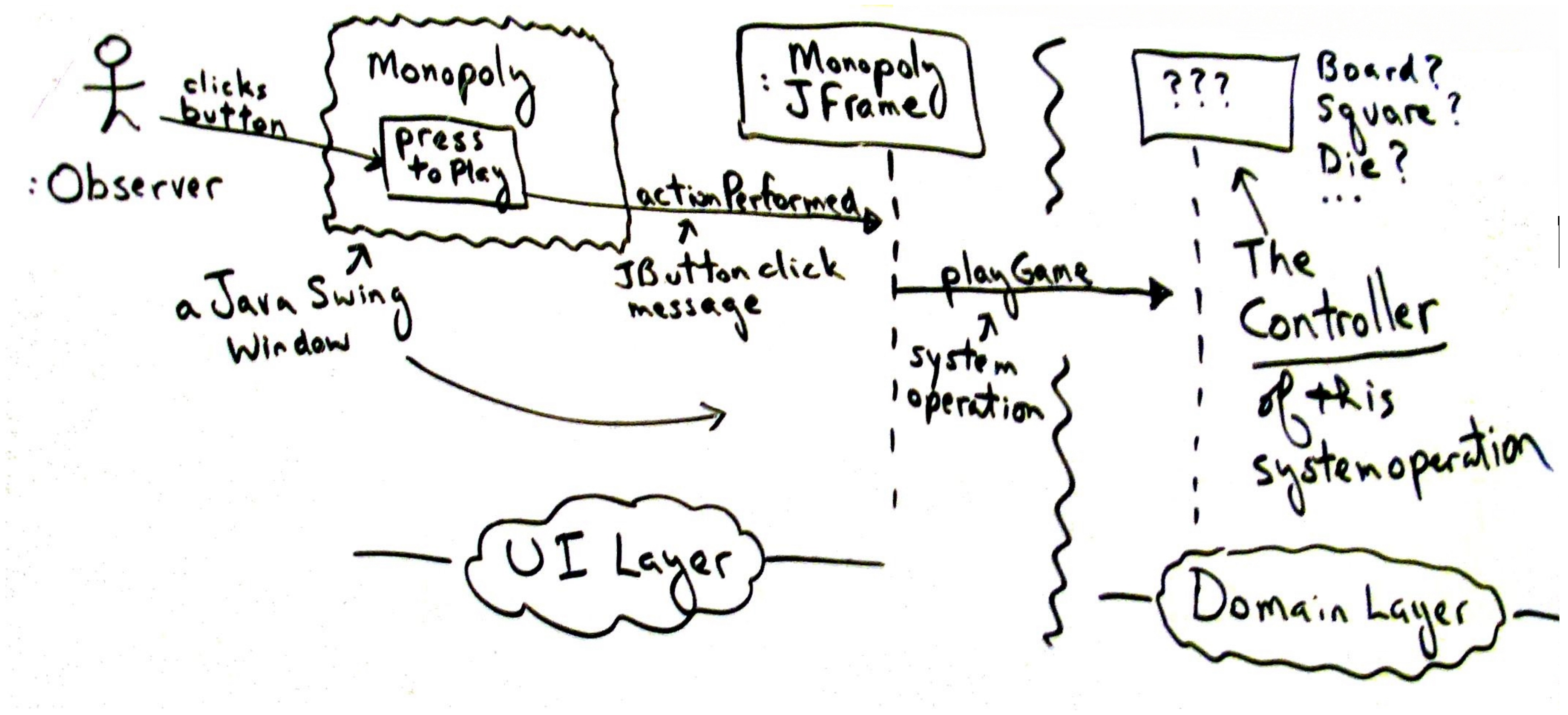
- Solution: Assign the responsibility to either…
  - A façade controller, representing the overall system and handling all system operations, or
  - A use case controller, that handles all system events for a single use case

Controller Example



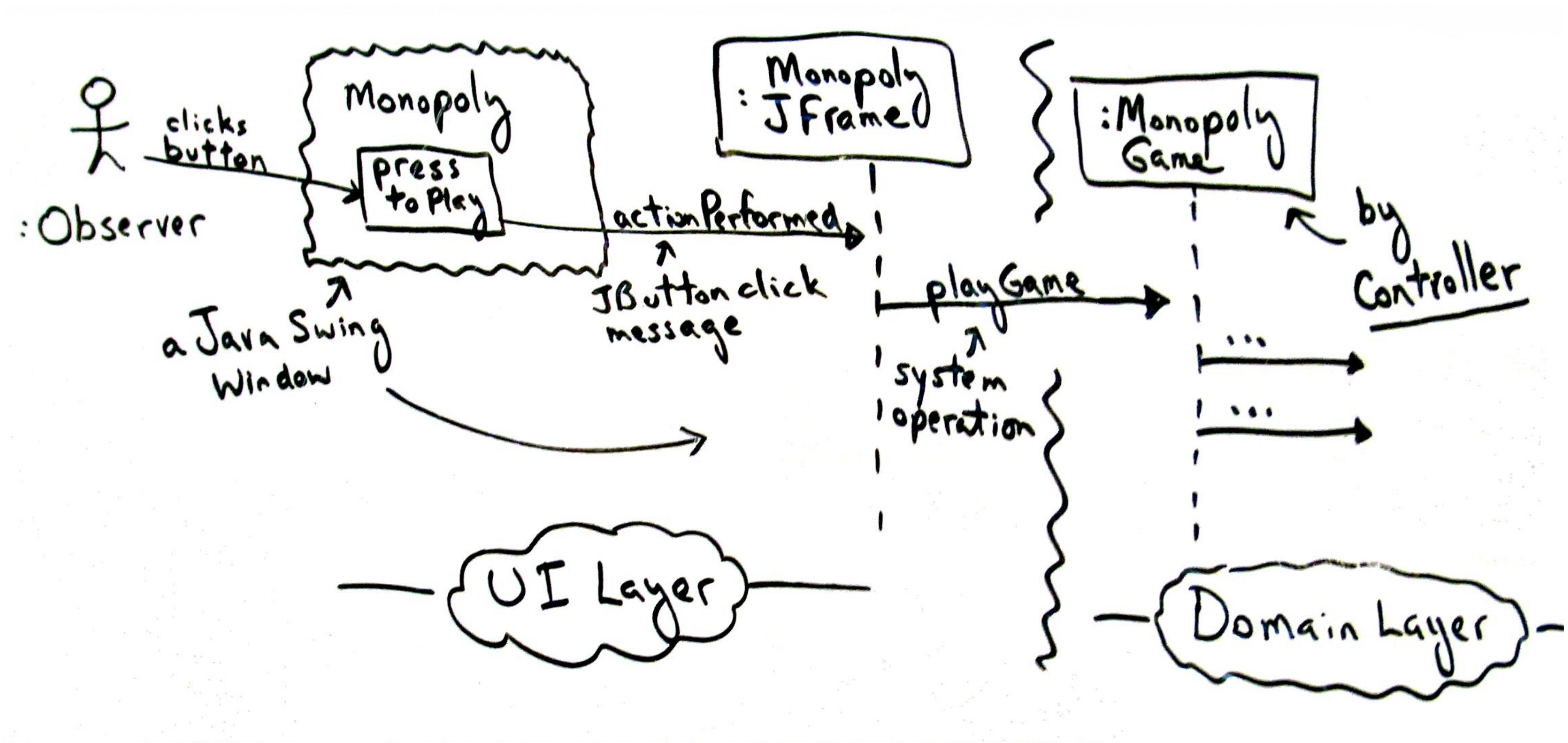
What Domain Layer class should own handling of the *enterItem* system operation?

Layered view of Monopoly Game



Who mediates between UI and Domain layers?

More on Monopoly



Let MonopolyGame be the controller …

Guidelines

- Controller should delegate to other domain layer objects
- Use façade controller when…
  - There are a limited number of system operations, or
  - When operations are coming in over a single "pipe"
- Use use case controller when a façade would be bloated (low cohesion!)

Controller Benefits and Issues

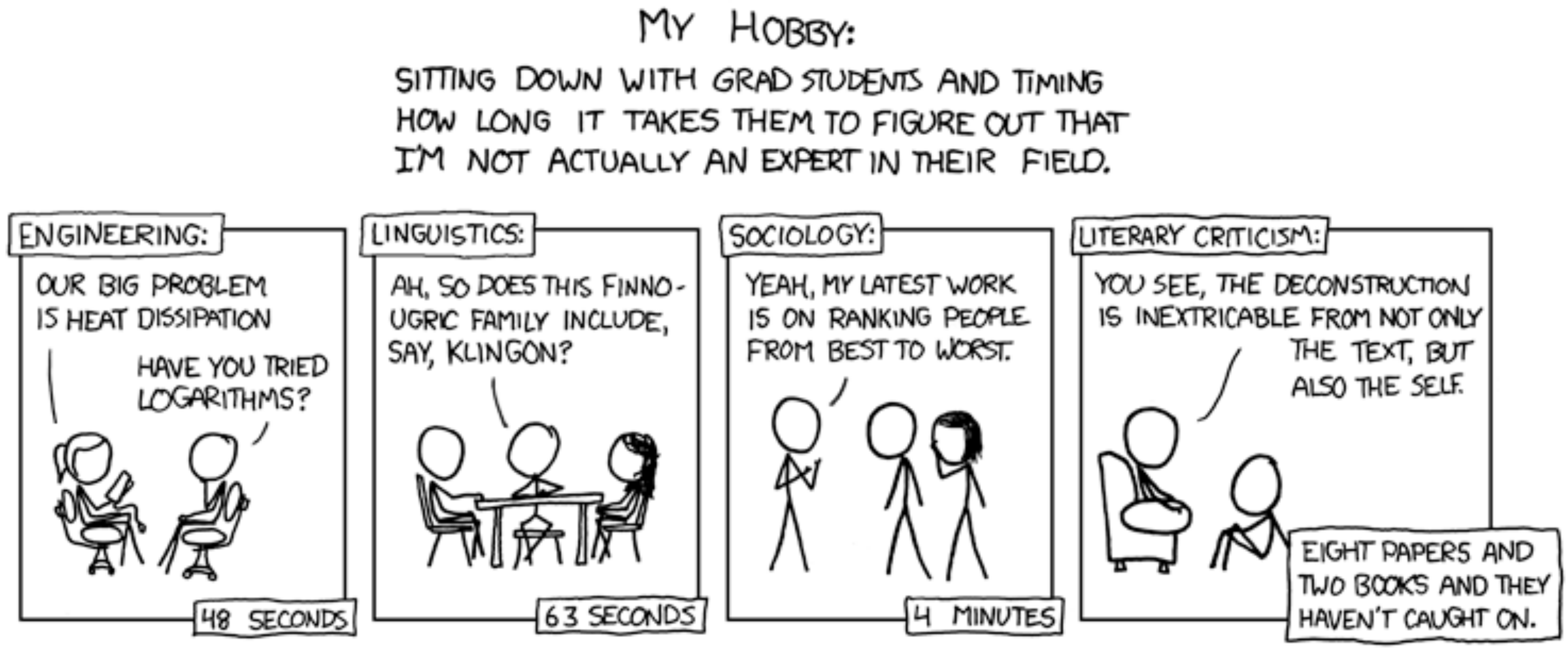
- Benefits:
  - Increased potential for reuse
  - Can reason/control the state of a use case
    - e.g., don't close sale until payment is accepted
- Issues:
  - Controller bloat—too many system operations
  - Controller fails to delegate tasks
  - Controller has many attributes

Switch from façade to use case controllers

Delegate!

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Imposter


MY HOBBY:
SITTING DOWN WITH GRAD STUDENTS AND TIMING HOW LONG IT TAKES THEM TO FIGURE OUT THAT I'M NOT ACTUALLY AN EXPERT IN THEIR FIELD.
ENGINEERING:
OUR BIG PROBLEM IS HEAT DISSIPATION
HAVE YOU TRIED LOGARITHMS?
48 SECONDS
LINGUISTICS:
AH, SO DOES THIS FINNO-UGRIC FAMILY INCLUDE, SAY, KLINGON?
63 SECONDS
SOCIOLOGY:
YEAH, MY LATEST WORK IS ON RANKING PEOPLE FROM BEST TO WORST.
4 MINUTES
LITERARY CRITICISM:
YOU SEE, THE DECONSTRUCTION IS INEXTRICABLE FROM NOT ONLY THE TEXT, BUT ALSO THE SELF.
EIGHT PAPERS AND TWO BOOKS AND THEY HAVEN'T CAUGHT ON.

If you think this is too hard on literary criticism, read the Wikipedia article on deconstruction.

Coupling

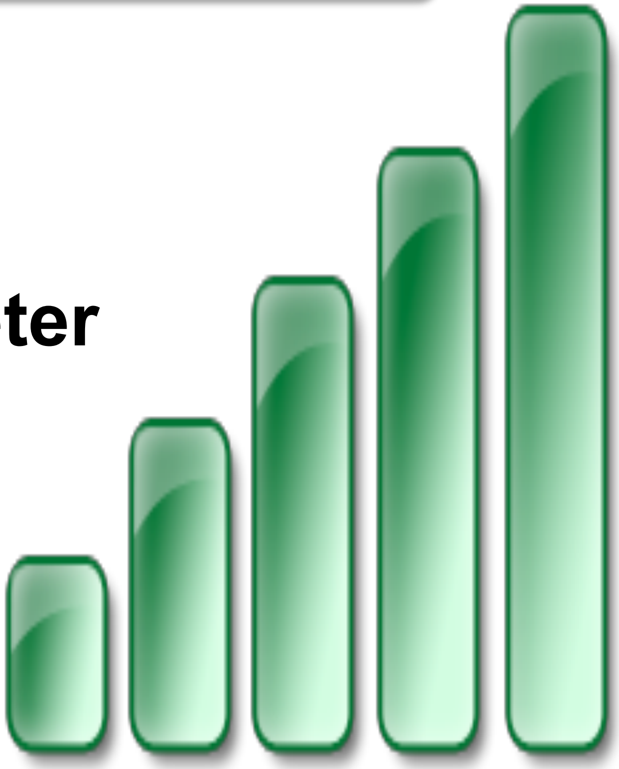
An Evaluative Principle

- A measure of how strongly one element:
  - is connected to,
  - has knowledge of, or
  - relies on other elements
- Goal: Low (or weak) coupling
- What are some problems with high coupling?

Common Couplings

Very strong coupling

- *A* is a subclass of *B*
- *A* implements an interface *B*
- *A* has a method with a parameter or variable of type *B*
- *A* calls a static method of *B*
- *A* has an attribute of type *B*



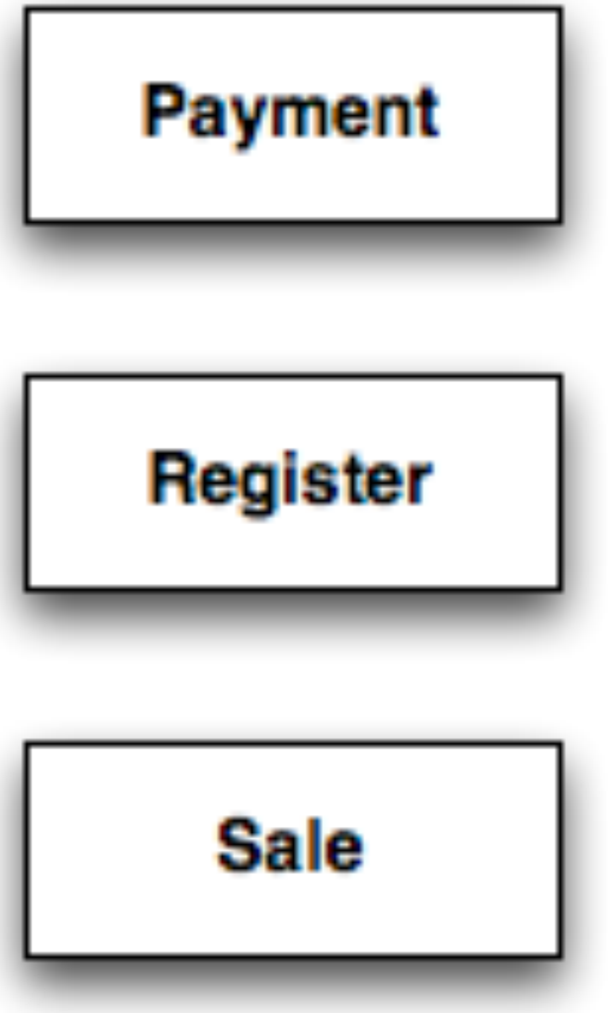
GRASP: Low Coupling

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

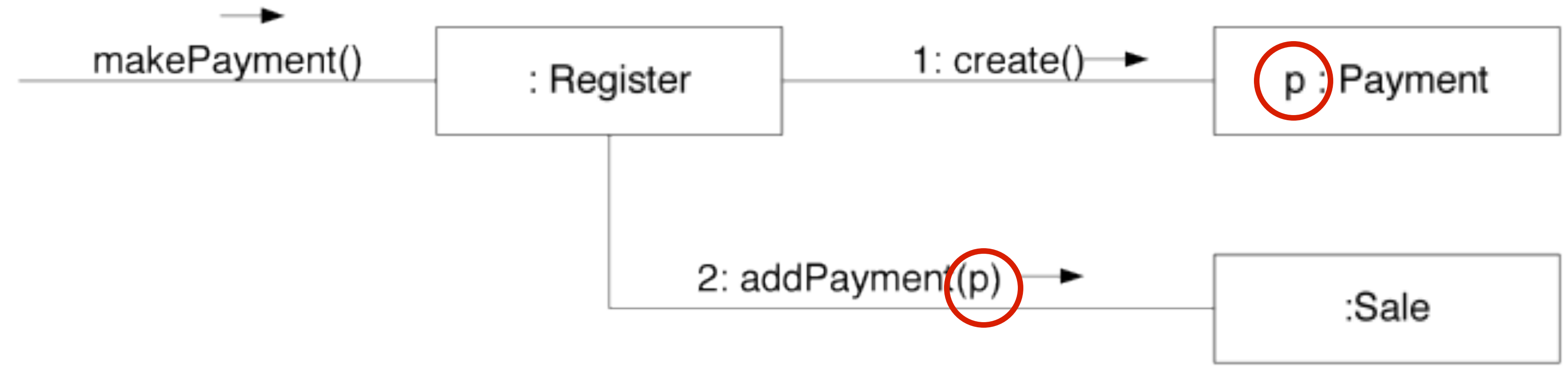
Solution: Assign a responsibility so that coupling remains low. Use this principle to evaluate alternatives.

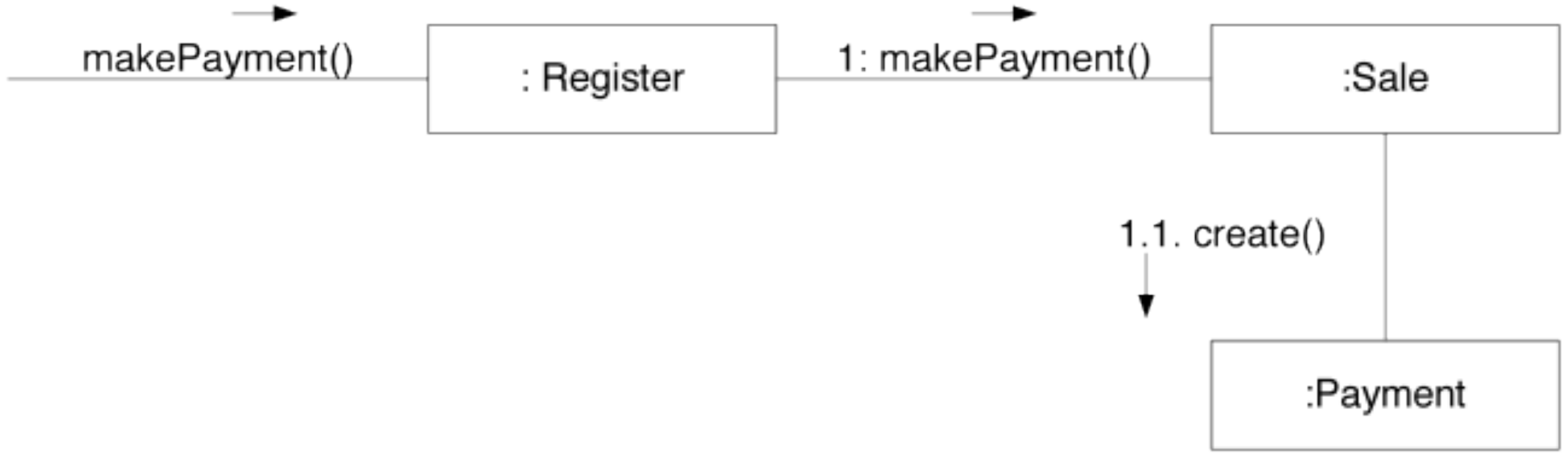
Low Coupling Example

- Suppose we need to create a *Payment* instance and associate it with a *Sale*
- Who should be responsible?



Which option has Lower Coupling?





Advice: Pick Your Battles

- Coupling to stable, pervasive elements isn't a problem
  - e.g., *java.util.ArrayList*
- Coupling to unstable elements can be a problem
  - Unstable interface, implementation, or presence
- Clearly can't eliminate coupling completely!

grasping hands ~ franziska lang

Cohesion

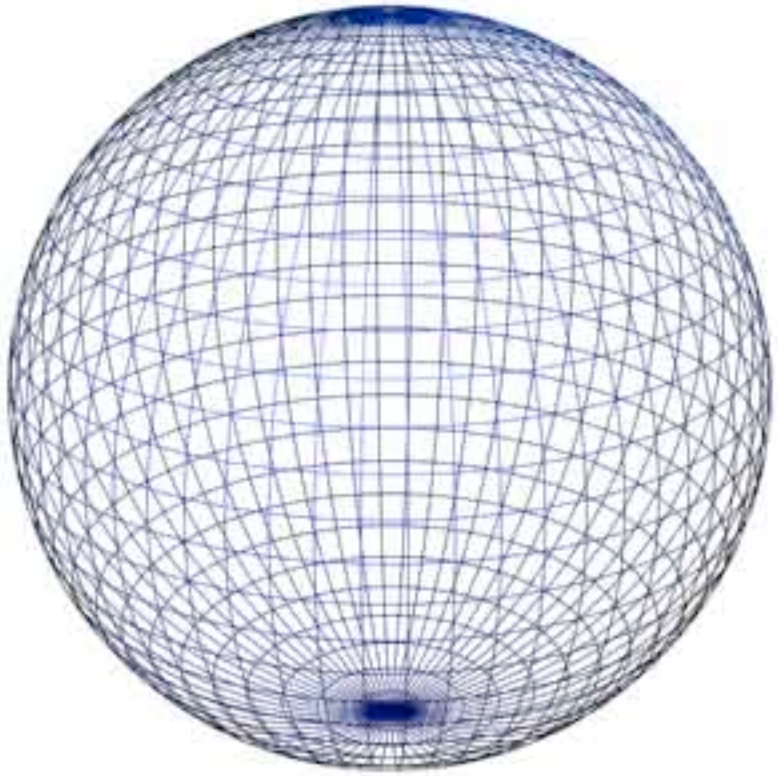
Another Evaluative Principle

- A measure of how strongly related and focused the responsibilities of a class (or method or package…) are
- Goal: high (strong) cohesion
- What are some problems with low cohesion?

GRASP: High Cohesion

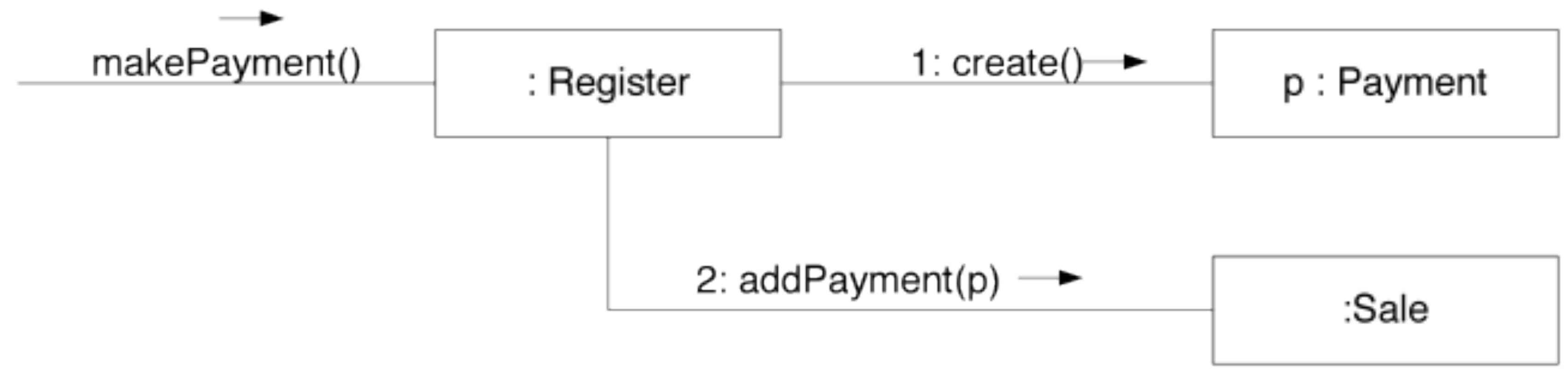
Problem: How do you keep objects focused, understandable, and manageable, and as a side-effect, support low coupling?

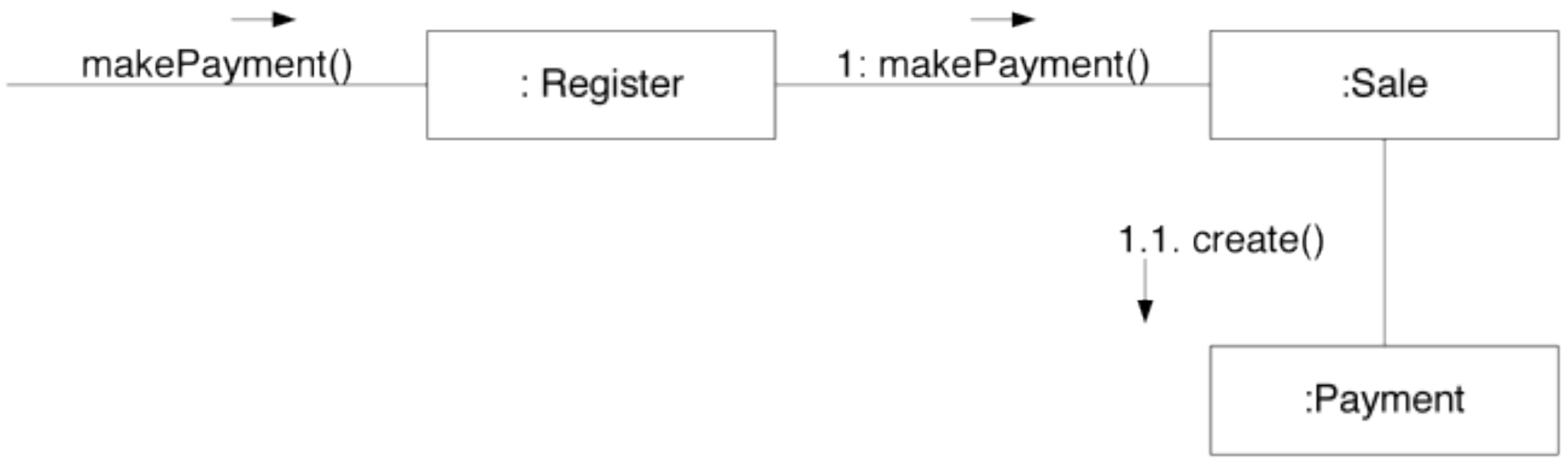
Solution: Assign a responsibility so that cohesion remains high. Use this principle to evaluate alternatives.



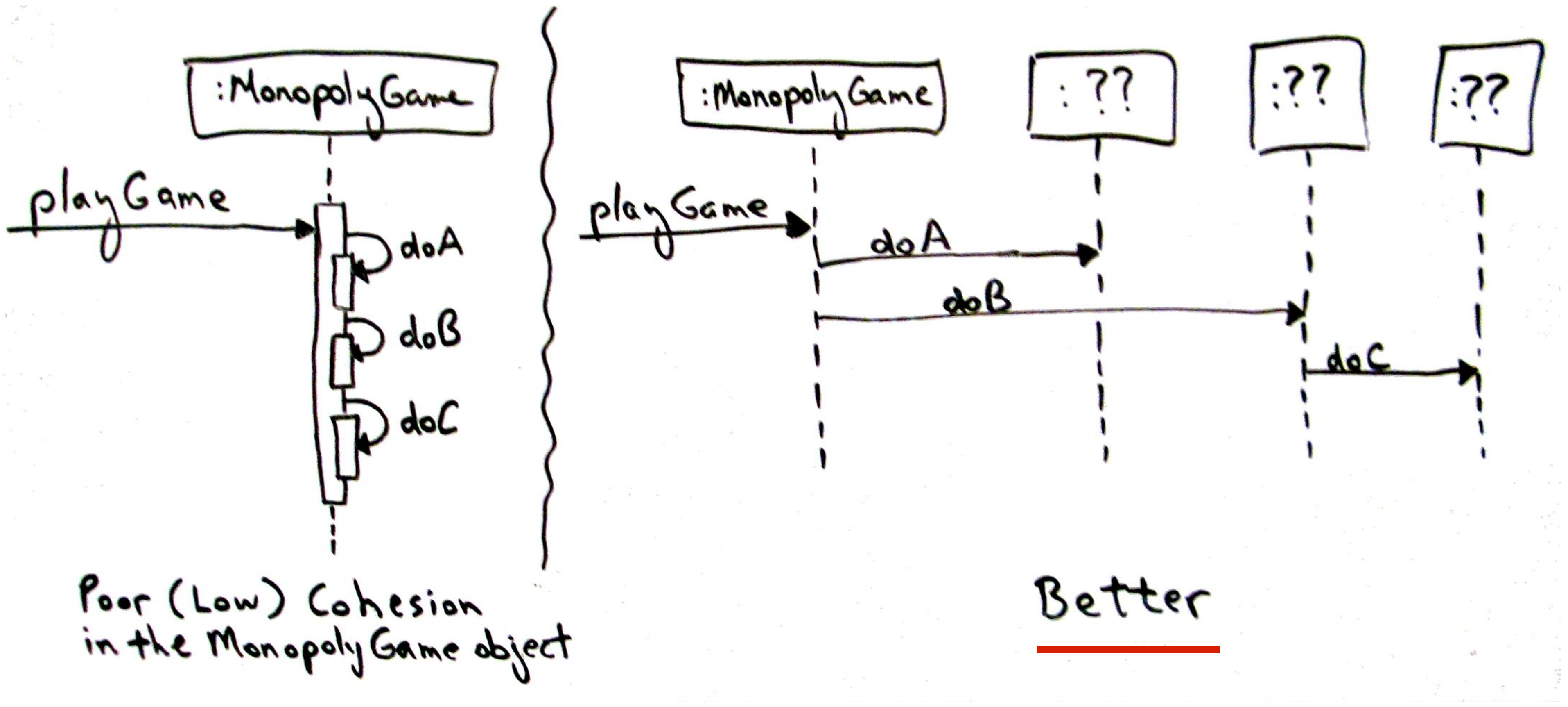
ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Which option has higher Cohesion?





Design Alternatives for High Cohesion



Cohesion Guidelines

A highly cohesive class has small number of highly related operations/methods

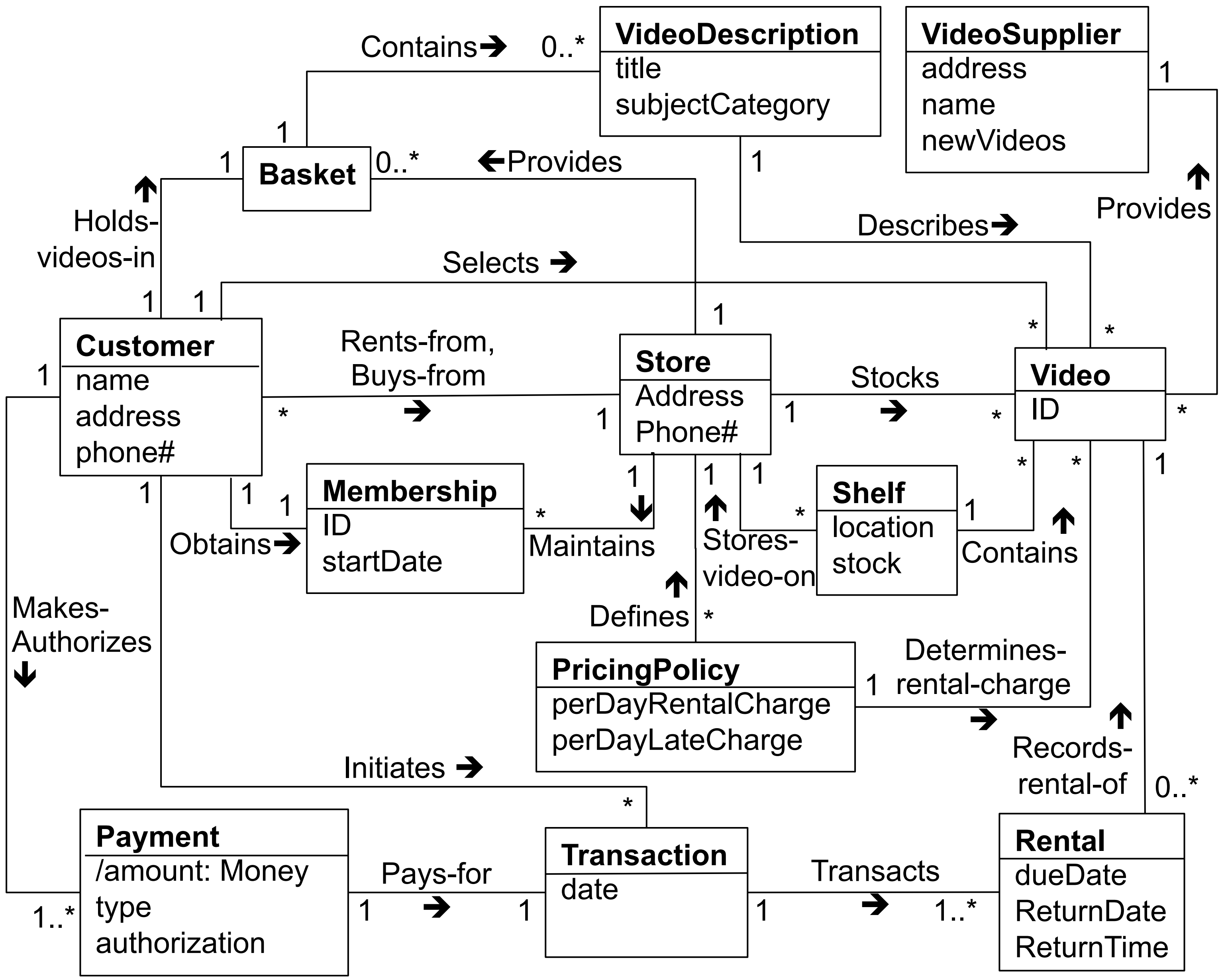
- Does not do "too much" work

- Inherent trade-offs of Cohesion and Coupling
  - To minimize coupling, a few objects have all responsibility
  - To maximize cohesion, a lot of objects have limited responsibility
  - Trade-off from alternative designs

Exercise on Creator Examples

- Break up into your project teams
- Given the following:
  - Domain Model for BBVS

1. Identify a couple potential Controller Patterns
2. Identify a couple potential Information Expert Patterns


VideoDescription
title
subjectCategory
VideoSupplier
address
name
newVideos
Contains→ 0..*
1
Basket
1
0..*
←Provides
1
1
Provides ↑
Describes→
Holds-videos-in ↑
Selects →
1 1
Customer
name
address
phone#
Rents-from, Buys-from →
Store
Address
Phone#
Stocks →
Video
ID
Membership
ID
startDate
Obtains→
Maintains ↓
Stores-video-on ↑
Shelf
location
stock
Contains ↑
Defines ↑
Makes-Authorizes ↓
PricingPolicy
perDayRentalCharge
perDayLateCharge
Determines-rental-charge →
Records-rental-of ↑
Initiates →
Payment
/amount: Money
type
authorization
Pays-for →
Transaction
date
Transacts →
Rental
dueDate
ReturnDate
ReturnTime
1..*
0..*

Homework and Milestone Reminders

- Read Chapter 18 on GRASP Examples

- Homework 3 – BBVS Logical Architecture and Preliminary Design
  - Due by 11:59pm Today, Tuesday, January 4th, 2011

- Milestone 3 – Junior Project SSDs, OCs, and Logical Architecture
  - Due by 11:59pm on Friday, January 7th, 2011

- Homework 4 – BBVS Design using GRASP
  - Due by 11:59pm Tuesday, January 11th, 2011