More GRASP'ing and Use Case Realization

Shawn Bohner
Office: Moench Room F212
Phone: (812) 877-8685
Email: bohner@rose-hulman.edu

GRASP

- General Responsibility Assignment Software Patterns (or Principles)

1. Low Coupling
2. High Cohesion
3. Information Expert
4. Creator
5. Controller

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

Q1

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

Q2

Low Coupling and High Cohesion

- Inherent trade-offs of Cohesion and Coupling
  - To minimize coupling, a few objects have most of the responsibility
  - To maximize cohesion, a lot of objects have limited responsibility
  - Trade-off from alternative designs for best results
- Support both by
  - Evaluating alternatives to keep objects focused, understandable, and maintainable
  - Assigning so object's responsibilities are closely related
  - Avoid spreading the responsible objects too thin
  - "Teamwork"

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Information Expert

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

Q3,4

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*

Q7,8

Team Creativity

Q11

Controller

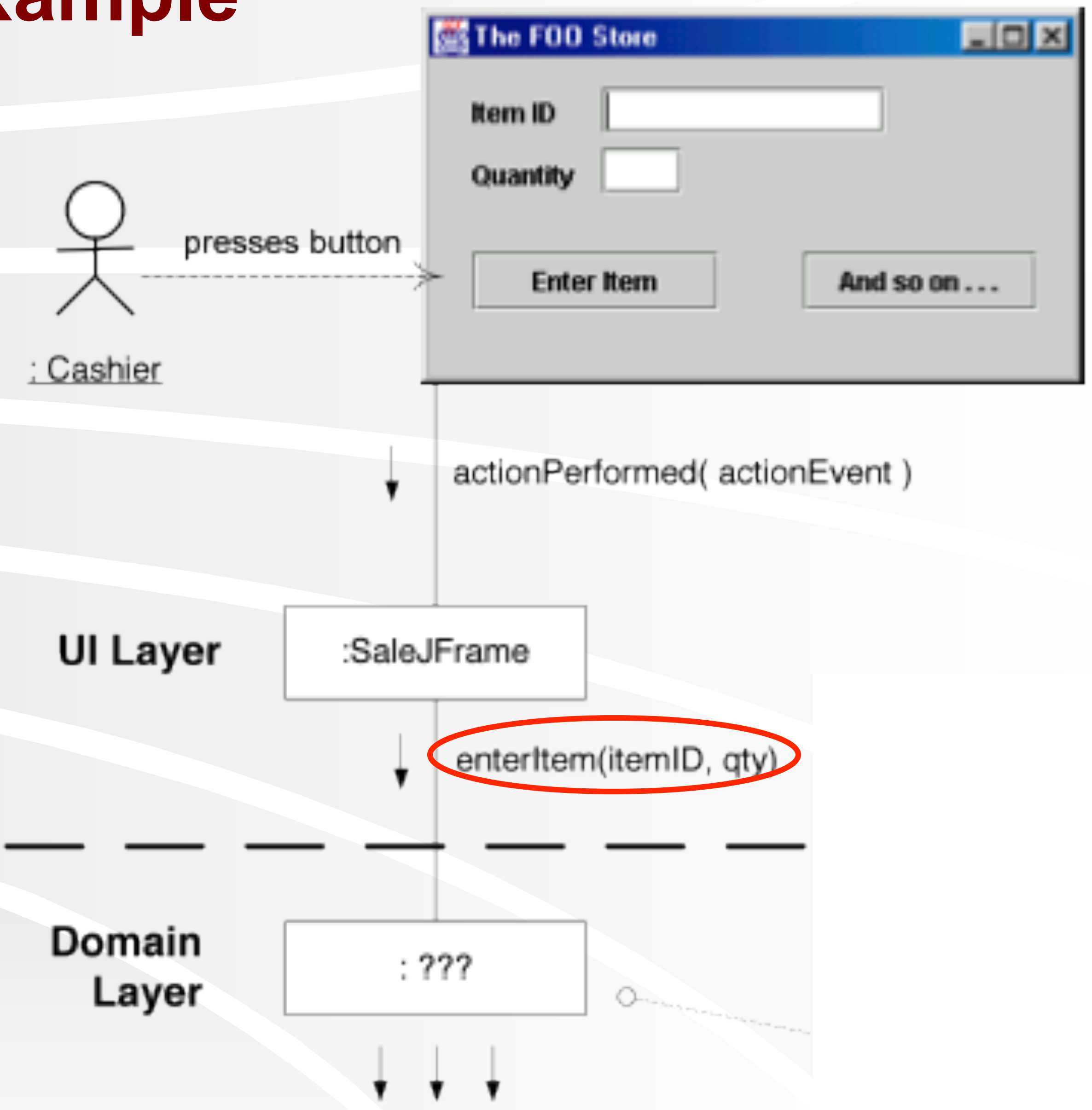
- Problem: What first object beyond the UI layer receives and coordinates a *system operation*

- 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

Q12,13

Controller Example

What domain layer class should own handling of the *enterItem* system operation?



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!)

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Controller Benefits

- Increased potential for reuse
- Can reason/control the state of a use case
  - e.g., don’t close sale until payment is accepted

Controller Issues

Switch from façade to use case controllers

- Controller bloat—too many system operations
- Controller fails to delegate tasks
- Controller has many attributes

Delegate!

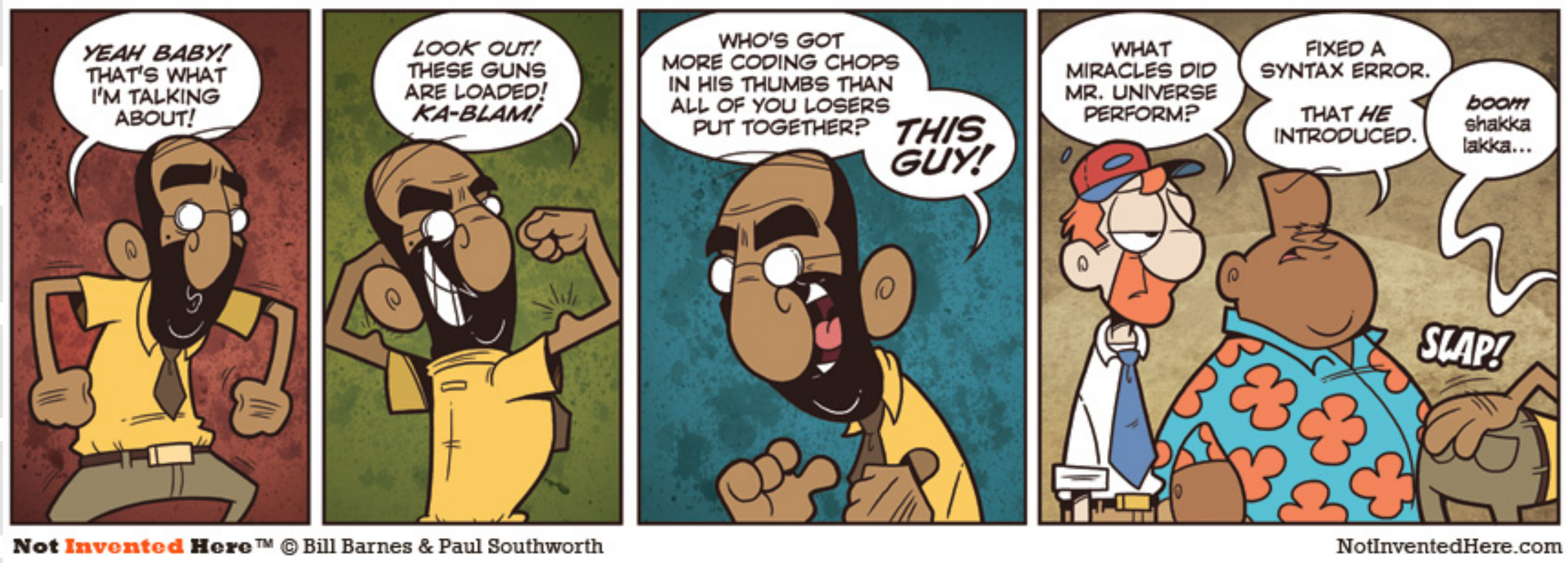
ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Team Control

Q14,15

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Cartoon of the Day


YEAH BABY! THAT'S WHAT I'M TALKING ABOUT!
LOOK OUT! THESE GUNS ARE LOADED! KA-BLAM!
WHO'S GOT MORE CODING CHOPS IN HIS THUMBS THAN ALL OF YOU LOSERS PUT TOGETHER?
THIS GUY!
WHAT MIRACLES DID MR. UNIVERSE PERFORM?
FIXED A SYNTAX ERROR.
THAT HE INTRODUCED.
boom shakka lakka...
SLAP!
Not Invented Here™ © Bill Barnes & Paul Southworth
NotInventedHere.com

Jan 4, 2010. Used by permission

Getting a GRASP on Design

- No ‘magic’ to assigning responsibilities
- If you don’t have a reason for placing a method in a class, it shouldn’t be there!
  - You should be able to say: ‘I placed method X in class Y based on GRASP Z’

Use Case Realization

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Use Case Realization

The process of generating the design model from use cases and other requirements artifacts

- Use Cases drove the development of
  - Domain Model
  - System Sequence Diagrams
  - Operation Contracts

System Sequence Diagrams

- Helped us identify system operations
- Use these to begin interaction diagrams
  - System operations are the starting messages
  - Starting messages are directed at controller objects

Q1

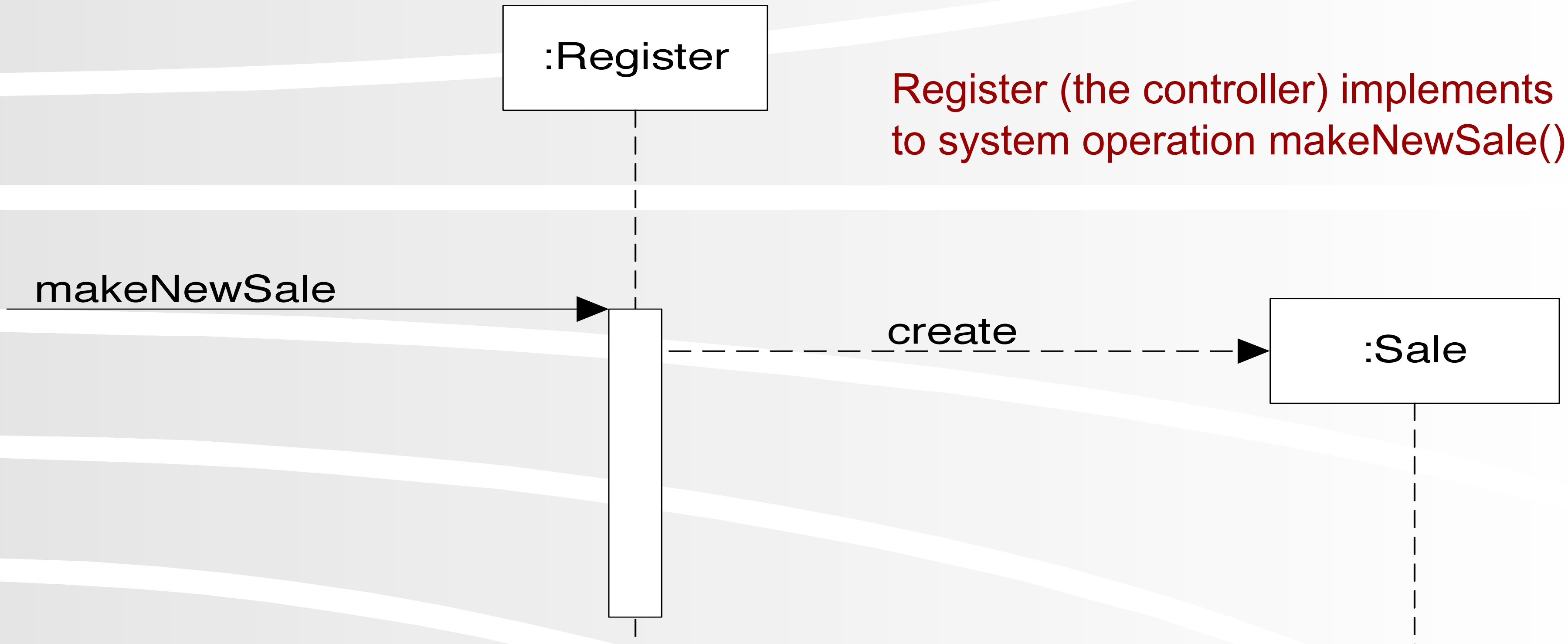
Operation Contracts

- Defined post-conditions of system operations as changes to objects/ associations in the domain model

- Use post-conditions to help determine…
  - What should happen in the interaction diagrams
  - What classes belong in the design class diagram

Also often discover classes that were missed in the domain model

Where to Begin



Register (the controller) implements to system operation makeNewSale()

- In code, you begin at the beginning
- In design, you defer design of the Start Up UC
  - Start Up handles created and initializing objects
  - Discover necessary objects as we do the other Ucs
  - So defer Start Up design to avoid rework

Q3

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Example: Design *makeNewSale*

Operation:	makeNewSale()
Cross References:	Use Case: Process Sale
Preconditions:	none
Postconditions:	◦ A *Sale* instance *s* was created ◦ *s* was associated with the *Register* ◦ Attributes of *s* were initialized

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Homework and Milestone Reminders

- Read Rest of Chapter 18 and Chapter 19
- Milestone 3 – Iteration 1: Junior Project
  - Finish Analysis Model (SSDs, OCs)
  - Logical Architecture - Package Diagrams, and
  - 1st (initial) Version of System
  - Due by 11:59pm on Friday, January 8th, 2009
- 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

ROSE-HULMAN INSTITUTE OF TECHNOLOGY