Introduction to GRASP: Assigning Responsibilities to Objects

CS 4354
Fall 2012

Jill Seaman

Object Design in the textbook

- Chapter 5 Analysis activities: from use cases to objects
  - ✦identified objects, associations, aggregations, attributes, inheritance relationships
  - ✦mapped use cases to objects with sequence diagrams,
  - ✦but didn't talk about designing operations of objects
- Chapter 9, Object design: Interface specification activities
  - ✦Identifying Missing Attributes and Operations
  - ✦still didn't talk about how to design the operations.

The design of behavior

- What methods in what classes? How should objects interact?
  - ✦These are critical questions in the design of behavior.
  - ✦Poor answers lead to abysmal, fragile systems with low reuse and high maintenance.
- Design of behavior implies assigning responsibilities to classes.
- Responsibilities:
  - ✦Knowing: storing information
  - ✦Doing: Calculating, coordinating, creating, …
- A message in a sequence diagram suggests a related responsibility.
- There are well-known best principles for assigning responsibilities.

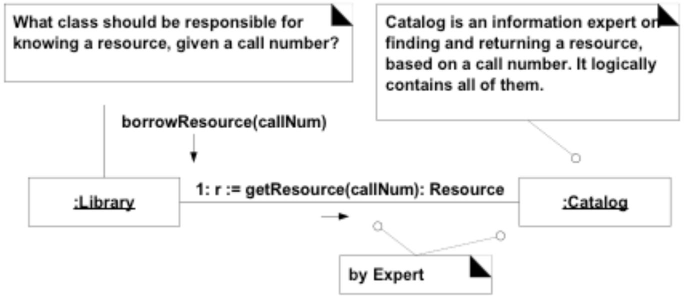
GRASP Patterns

GRASP

- Acronym for General Responsibility Assignment Software Patterns.
- Has nine core principles that object-oriented designers apply when assigning responsibilities to classes and designing message interactions.
  - ✦We will look at 5 of these 9 principles
- Can be applied during the creation of sequence diagrams.

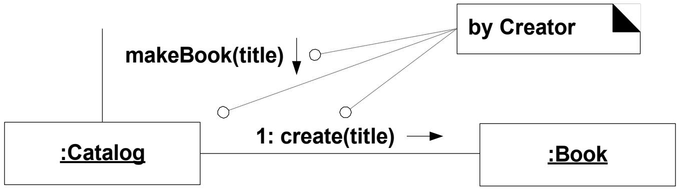
Pattern: Information Expert

- What is most basic, general principle of responsibility assignment?
- Assign a responsibility to the object that has the information necessary to fulfill it.
  - ✦"That which has the information, does the work."


What class should be responsible for knowing a resource, given a call number?
Catalog is an information expert on finding and returning a resource, based on a call number. It logically contains all of them.
borrowResource(callNum)
:Library
1: r := getResource(callNum): Resource
:Catalog
by Expert

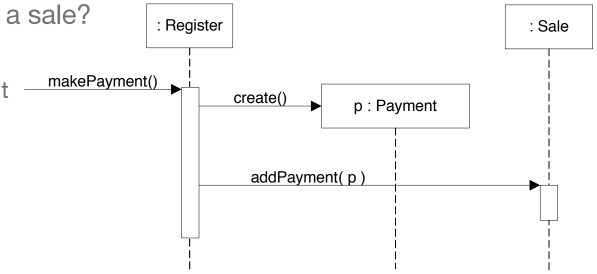
Pattern: Creator

- What object creates an X?
- Choose an object C, such that:
  - ✦C contains or aggregates X
  - ✦C closely uses X
  - ✦C has the initializing data for X
- The more, the better.



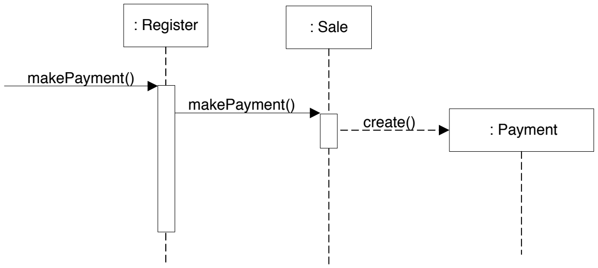
Pattern: Low Coupling

- **Coupling** (in a class diagram) is a measure of how strongly one class is connected to, has knowledge of, or relies on other classes.
- How can our design provide greater independence, less vulnerability to change, and increased potential for reuse?
  - ✦Assign responsibilities in a way that promotes low coupling.
- Which class should be responsible for creating a Payment and associating it with a sale?
  - ✦Since Register records a payment IRL, it could be Register, by the Creator pattern:



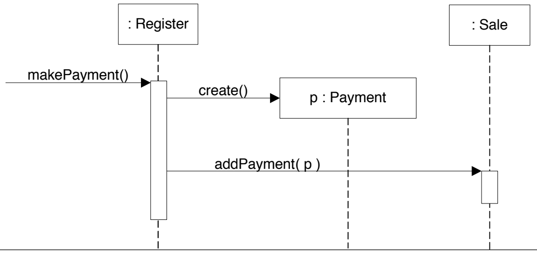
Pattern: Low Coupling

- In the previous example, Register is coupled to the Payment class.
- In the following example, the Sale has the responsibility of creating the payment
  - ✦This version has lower coupling because the Register doesn't need to know about the Payment class.



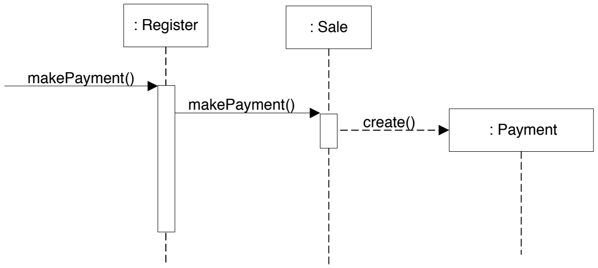
Pattern: High Cohesion

- **Cohesion** (in a class diagram) is a measure of how strongly related and focused the responsibilities of a class are.
- A class with low cohesion does many unrelated things, or does too much work. They are hard to understand, reuse, and maintain.
- How can our design keep complexity manageable?
  - ✦Assign responsibilities in a way that promotes high cohesion.
- Let's compare the same two examples as before with respect to cohesion:



Pattern: High Cohesion

- In the previous example, Register is responsible for creating a payment AND adding a payment to a sale.
- This is ok, but not if we keep piling responsibilities onto it.
- In the following example, no class has too much work (good delegation):

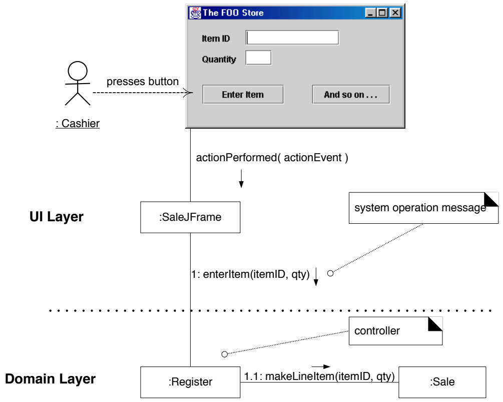


Pattern: Controller

- What class should handle system event messages (such as input from the user)?
- Solution: Choose a class whose name/job suggests:
  - ✦The overall "system," device, or subsystem (a kind of Façade class)
  - ✦OR, represents the use case scenario or session
- Recall: during analysis,we identified three types of objects:
  - ✦Entity Objects: persistent information tracked by system (domain objects)
  - ✦Boundary Objects: represent the interface between the actors and the system
  - ✦Control Objects: are in charge of realizing use cases
- Recall: MVC architectural pattern: the Controller component

Pattern: Controller

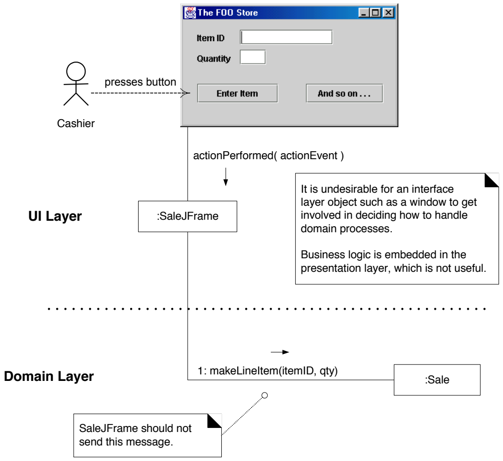
- In this example, the Register object (a controller) handles the input event.



Pattern: Controller

- In this example, SaleJFrame, a UI (boundary) object handles the input event

Don’t want the UI objects tightly coupled with the entity objects



It is undesirable for an interface layer object such as a window to get involved in deciding how to handle domain processes.

Business logic is embedded in the presentation layer, which is not useful.

SaleJFrame should not send this message.

Summary of Introduction to GRASP

- 5 principles for deciding how to assign responsibility (behavior) to classes:
  - ✦Information Expert
  - ✦Creator
  - ✦Low Coupling
  - ✦High Cohesion
  - ✦Controller
- These decisions are made during analysis and/or object design.
- These decisions are made (initially) when designing the interactive (sequence) diagrams from the use cases (deciding which messages are handled by which objects)