

# Lecture 10: Responsibilities

Kenneth M. Anderson

Object-Oriented Analysis and Design

CSCI 6448 - Spring Semester, 2005



## Goals for this Lecture

- Review content of Chapter 4 from the textbook
- Discuss Responsibilities
  - What are they?
  - How do we find them?
  - How do we describe them?
  - How do we assign them?



# Responsibilities

- The core of RDD is assigning responsibilities to objects
  - So, what is a responsibility?
- Responsibilities are general statements about software objects; they include
  - The actions an object performs
  - The knowledge an object maintains
  - Major decisions an object makes that affect others

# Example: Teakettle (I)

- Consider the design of a teakettle
  - What is the right form for a teakettle?
    - A teakettle holds water that can be heated until boiling
    - People can safely pick up a teakettle when it is filled with boiling water and pour a cup of tea
    - By convention, a teakettle whistles when the water boils
  - These characteristics can be restated as responsibilities
    - Pour contents without spilling or splashing
    - Hold water that can be heated until boiling
    - Is safe to hold and carry while water is hot
    - Notify when boiling occurs

## Example: Teakettle (II)

- ❦ Did we get this right? It depends on the boundaries we have set for the problem; in conventional terms we have the bases covered
- ❦ But some designers like to redefine the problem: “Its not the teakettle that needs to be designed, but the method of heating the water!”
- ❦ Here the teakettle becomes part of the context, rather than the “form being designed”
- ❦ This type of thinking might lead to innovation such as an “instant hot” unit that heats tap water as it flows through it
- ❦ The trick is to know when to indulge this type of thinking; sometimes it leads to innovation but sometimes it adds unnecessary complexity and expense
- ❦ For most people, the conventional teakettle works just fine!

## Finding Responsibilities

- ❦ Use Cases
  - ❦ Identify system responsibilities stated or implied by use cases
  - ❦ plug gaps in use cases by developing lower-level responsibilities
- ❦ Themes and Design Stories
- ❦ Follow “what if...then...and how” chains
- ❦ Identify stereotypical responsibilities
- ❦ Identify responsibilities to support relationships between candidates
- ❦ Patterns (!)

## Use Cases and Responsibilities

- ❊ Use cases describe our software from the perspective of an outside user
  - ❊ They don't tell how something is accomplished
  - ❊ We need to “bridge this gap” by transforming these descriptions into explicit statements about actions, information, or decision-making responsibilities
- ❊ Bridging the Gap
  - ❊ Identify things the system does and the information it manages
  - ❊ Restate these things as responsibilities
  - ❊ Break them down into smaller parts if necessary and assign them to appropriate objects

## Example: University Enrollment

- ❊ A student can register online for classes by filling out and submitting an online registration form for approval. While filling out the registration form, a student can browse course schedules, cross-listed courses, audit degree requirements, and update personal and financial aid information. The student can also access the “waitlist class” and “drop class” functions
- ❊ The system should identify problems as courses are added, such as time conflicts, full classes, lack of prerequisites, etc.

## Example: Responsibilities

- ❁ Generate and display an online registration form (something needs to know the structure of the form and how to display it)
- ❁ Provide feedback as the student enters course selections about conflicts or problems (Something needs to check that a student can sign up for a course; a component is also needed to display feedback about the results)
- ❁ Provide capabilities for browsing, auditing degree requirements, and updating personal/financial information (browsing sounds like a big responsibility, auditing sounds like a complex process, updating personal information will require specific boundary, controller, and domain classes)
- ❁ ...

## Example: Specific Scenario

- ❁ 1. Student logs in
- ❁ 2. System verifies that student is eligible to register and displays reg. form
- ❁ 3. Student adds courses to schedule
- ❁ 4. System verifies schedule and returns approved courses for confirmation
- ❁ 5. Student confirms schedule
- ❁ 6. System updates course rosters and confirms successful registration

## Example: More responsibilities

- Check that student is eligible to register
  - From step 2
- Add student to course rosters
  - From step 6
- Display confirmation of registration
  - From step 6
- Validate each course in schedule meets constraints such as prerequisites, etc.
  - From step 4

## Example: Filling in gaps

- These directly derived responsibilities have gaps; ask questions and identify additional responsibilities
  - How are prerequisites specified?
    - A relationship between course objects?
      - Possibly need structurer to handle this
  - What states does a student's schedule go through? "build/submit/confirm"
    - Who manages this life cycle? The schedule object?
  - Does registering happen in "real time"?
  - How much help should the system give to a student when things go wrong?

# Themes and Design Stories

- In lecture 9, we saw that design stories can be used to identify candidate objects... they can also be used to identify system responsibilities
- Lets return to the example design story that discussed an Internet banking application (See page 117-119)

# “What if” scenarios

- Asking “what if” questions can lead to lines of reasoning that identify additional responsibilities
  - What if the database goes down before my schedule is confirmed?
    - Is the student out of luck? Can the schedule be saved elsewhere and retrieved for later submission?
- This type of thinking will lead to new candidates with responsibilities to handle this situation

# Stereotypical Responsibilities

- As mentioned before, stereotypes have common sets of responsibilities that can help generate specific responsibilities for objects that play these roles
  - Information holders “know” things
  - Service providers “do” things
  - Structurers “create” and “maintain” things
  - ...

# Responsibilities from Relationships

- A meeting has attendees
  - Who has the following responsibility:
    - “How many people attended this meeting?”
  - Probably the meeting object
    - This responsibility was derived from the relationship between the two objects however



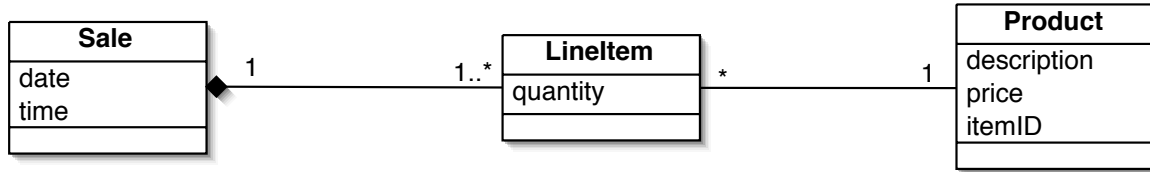
## Patterns for Identifying Responsibilities

- Craig Larman has developed patterns for helping to identify responsibilities (we will review four here; he has actually developed nine such patterns)
  - Information Expert (or Expert)
  - Creator
  - Low Coupling
  - High Cohesion
- This material taken from Craig Larman's Applying UML and Patterns. © Craig Larman, 2002 ISBN 0-13-092569-1

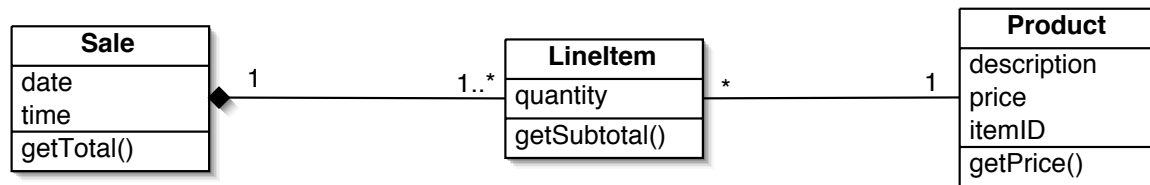
## Information Expert

- Assign a responsibility to the class that has the information necessary to fulfill it
  - Consider a “cash register” domain with the following objects: Sale, LineItem, Product
  - Consider the responsibility: “Know the grand total of a Sale”
  - It seems obvious that the sale object should have this responsibility, but let's look at the implications

# Sales Example: Class Diagram



If we want to get the total value of a sale, we would need to call a method like `getTotal()` on `Sale`; This method would need to call a method like `getSubtotal()` on `Lineltem`, since `Lineltem` is the “expert” for this information; But this method would need to call `getPrice()` on `Product` since only `Product` has this information



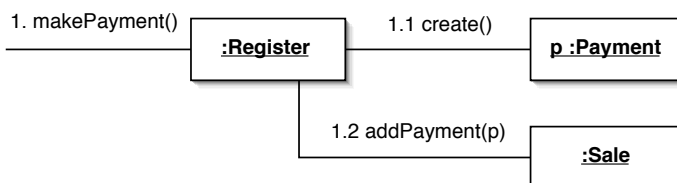
# Creator

- ❁ Assign class B the responsibility of creating an instance of A if one or more of the following is true
  - ❁ B aggregates A
  - ❁ B contains A
  - ❁ B records instances of A
  - ❁ B closely uses A
  - ❁ B has the data required to initialize A
- ❁ In our previous example, `Sale` should be assigned the responsibility of creating `Lineltem` objects; this means that `Sale` will need a method like “`addLineltem()`” or similar

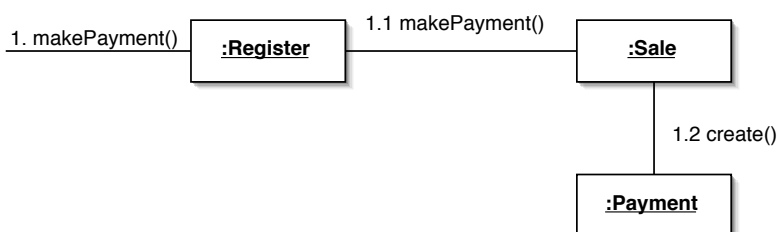
# Low Coupling

- ❁ Assign a responsibility so that coupling remains low
  - ❁ coupling is a measure of how strongly a class is connected to, has knowledge of, or relies on other classes
  - ❁ Building on our “cash register” example, consider the classes Payment, Register, and Sale
  - ❁ How should we handle the “make payment” responsibility

# Two Options



Note that we name the Payment object above, so we can pass it as a parameter.



Which option should we choose?

The first requires the Register object to know about two objects

The second requires Register to know about only one object

All things being equal, this pattern would choose option 2

# High Cohesion

- Assign a responsibility so that cohesion remains high
  - In terms of object design, cohesion is a measure of how strongly related and focused the responsibilities are of a class
- In previous example, this pattern would pick option 2 again; The Register object is likely to have many operations that it must handle (or coordinate); if it has to know the details of handling each operation it will lack cohesion

# Recording Responsibilities

- Responsibilities should be recorded on CRC cards
- If you can't find a "home" for a responsibility; record it on a post-it note and place it to one side...eventually a home will be found for it
  - or it may need to be decomposed into smaller responsibilities that are easier to assign
- You may also add unassigned responsibilities to a list
  - as new candidates are developed you can pull responsibilities from the list (or the list might drive the creation of new candidates!)

# Strategies for Assigning Responsibilities

- So far, we have discussed how to search for responsibilities
- Now we will look at techniques for assigning responsibilities to candidates
- The key is looking for “important” objects, such as
  - objects that bridge layers in your application or coordinate the actions of other objects
  - objects that are visible across multiple object neighborhoods
  - objects that represent key domain concepts
  - objects that manage complex services

# Initial Assignments

- Choose objects that you deem central players and concentrate on them first
  - Expect to work on more than one object at a time
- You can cluster candidates to work on by
  - domain concepts
  - use cases
  - design themes
  - interfacers

# Techniques

- State responsibilities generically; aim for a level above individual attributes or operations
- For a customer object, say
  - “Knows name and preferred ways of being addressed”
- Don’t say
  - “Knows first name”
  - “Knows last name”
  - “Knows nick name”

# Techniques, continued

- Use the right level of description
  - Responsibilities should be worded to match the level of abstraction of their associated candidates
    - The book provides two examples on pages 129-131
    - The MVC pattern example contains broadly stated responsibilities because they describe roles that can be adopted by many different types of objects
    - The Document example contains terse descriptions that are more specific to their associated objects

# Techniques, continued

## ☘ Use strong descriptions

### ☘ Vague responsibilities do not help

#### ☘ So use verbs like

☘ remove, merge, calculate, activate

#### ☘ rather than

☘ organize, record, find, process, maintain

## ☘ Avoid nonessential (out-of-scope) responsibilities

## ☘ Do not overlap responsibilities

☘ For instance do not have a client verify the data it sends AND have the server verify the data it receives; have the server verify and the client be able to handle situations where data is rejected

# Techniques, continued

## ☘ Handle large or complex responsibilities by dividing them or sharing them among multiple objects

## ☘ Balance responsibilities to create “intelligent” objects

☘ Make sure an object isn't doing too much

☘ Keep behavior with related information

☘ Keep information about one thing in one place

☘ Distribute system intelligence

# Testing Candidate Quality

- Once responsibilities have been assigned, check to see that each candidate is well formed
  - Does it stick to its purpose?
  - Are its responsibilities clearly stated?
  - Do its responsibilities match its role?
  - Is it of value to other objects in its neighborhood?
- What's Next?
  - Designing collaborations