CS 619 Introduction to OO Design and Development

Domain Models (Part 2)

Fall 2012

Responsibility and Collaborations

What are responsibilities?

- The public services an object may provide to other objects,
  - the knowledge an object maintains and provides
  - the actions it can perform
- ... not the way in which those services may be implemented
  - specify what an object does, not how it does it
  - don’t describe the interface yet, only conceptual responsibilities

Responsibility and Collaborations

What are collaborations?

- other objects necessary to fulfill a responsibility
  - when collaborating these other objects play a role
  - to play this role, other objects must have certain responsibilities
- empty collaborations are possible

Identify Responsibilities and the associated collaborations

- Scenarios and Role Play
  - Perform scenario walk-throughs of the system where different persons
  - “play” the classes, thinking aloud about how they will delegate to other objects.
- Verb phrase identification.
  - Similar to noun phrase identification, except verb phrases are candidate responsibilities.
- Class enumeration.
  - Enumerate all candidate classes and come up with an initial set of responsibilities.

Role-playing with CRC cards

Role-playing is a way to achieve common understanding between all parties involved (domain experts, analysts, ...)

Basic Steps

1. **Create list of scenarios**
2. **Assign Roles**; Each member receives a number of (class-responsibility-collaborator) CRC Cards.
3. **Repeat**
   - 3.1 Rehearse Scenarios
     - Script = Responsibilities on CRC Cards
   - 3.2 Correct CRC Cards and revise scenarios
     - Rehearsals will make clear which parts are confusing
   - 3.3 Until scenarios are clear

CRC Card format (Front only)

Class Name:

Responsibilities:	Collaborators:
(what class does or knows)	*(which classes help it perform each responsibility)*

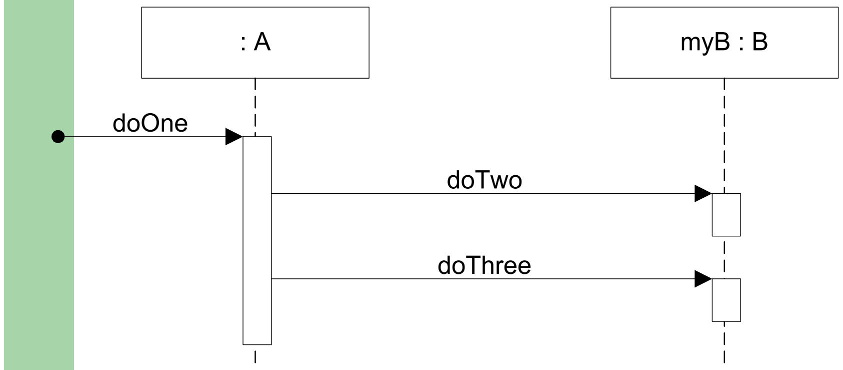
OOA with CRC cards

Step 1. Create a list of scenarios, bound the scope of design

- Directly from Use Case document
  - Choose a coherent set of use cases
- System Sequence Diagram (SSD)
  - Generated as part of requirement document
  - Use UML sequence diagram notation

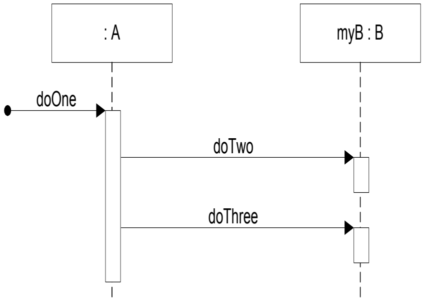
UML Sequence Diagrams

- Sequence diagrams illustrate interactions in a kind of fence format, in which each new object is added to the right:



UML Sequence Diagrams

```
public class A
{
   private B myB = new B();
   public void doOne()
  {
      myB.doTwo();
      myB.doThree();
  } // …
}
```



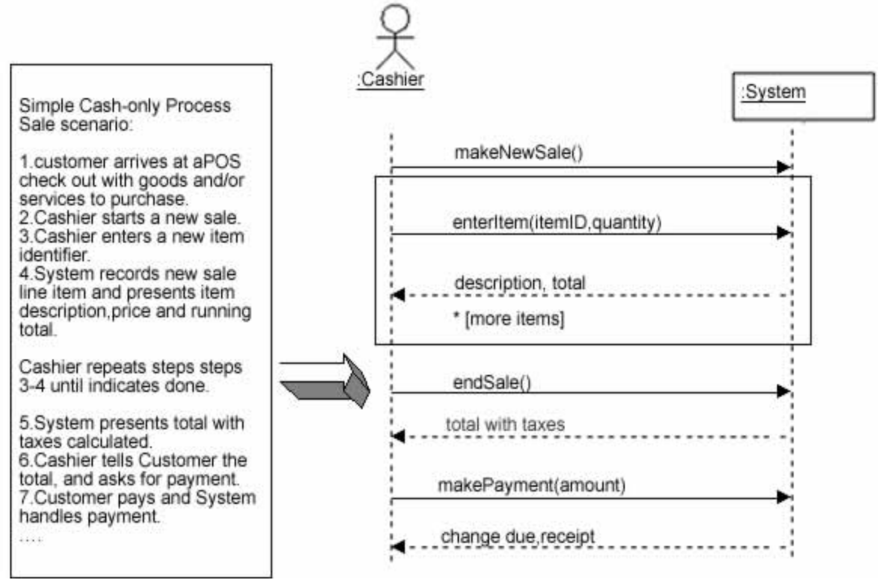
System Sequence Diagram (SSD)

For a use case scenario, an SSD shows:

- The System (as a black box) :System
- The external actors that interact with System
- The System events that the actors generate
- SSD shows operations of the System in response to events, in temporal order

- Develop SSDs for the main success scenario of a selected use case, then frequent and salient alternative scenarios

Example: use cases to SSD



Simple Cash-only Process Sale scenario:

1.customer arrives at aPOS check out with goods and/or services to purchase.
2.Cashier starts a new sale.
3.Cashier enters a new item identifier.
4.System records new sale line item and presents item description,price and running total.

Cashier repeats steps steps 3-4 until indicates done.

5.System presents total with taxes calculated.
6.Cashier tells Customer the total, and asks for payment.
7.Customer pays and System handles payment.
....

From Use Case to SSD

How to construct an SSD from a use case:

1. Draw System as black box on right side
2. For each actor that directly operates on the System, draw a stick figure and a lifeline.
3. For each System events that each actor generates in use case, draw a message.
4. Optionally, include use case text to left of diagram.

OOA with CRC cards

Step 2. Assign Roles:

- List the candidate class/objects
- Difference individuals are different objects.

Step 3. Start Role Playing:

- Execute the scenarios one by one
  - add new classes/objects/components as needed
  - add new responsibilities
  - delegate to other classes / persons

Library Example:

- Problem Description

Library system for checking out, returning, and searching for books. No more than 5 books can be loaned by one borrower at a time. And if a book is returned after its overdue date, a fine has to be paid.

Library Example: Detailed Use Case

Name: Check Out Book

Description: User checks out a book from the library

Actor: User

Main scenario:

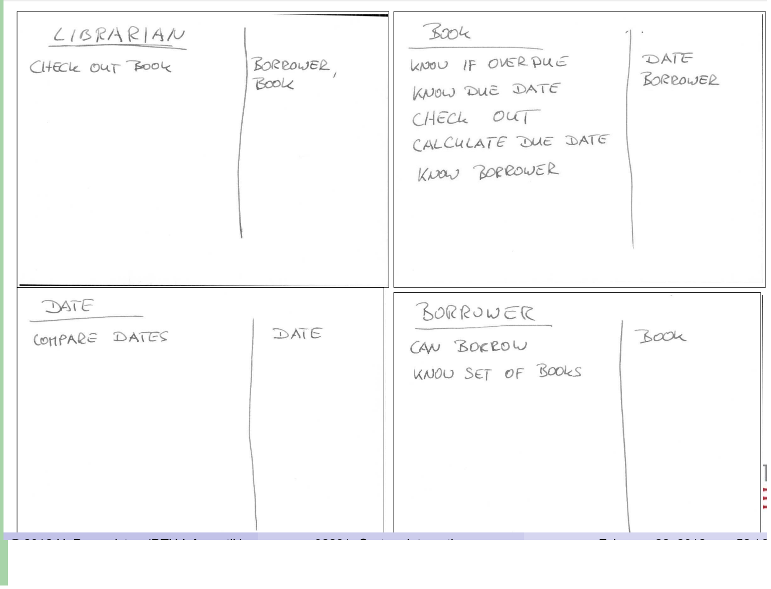
- 1 A user presents a book for check-out at the check-out counter
- 2 The system registers the loan

Alternative scenarios:

- The user already has 5 books borrowed
  - 2a The system denies the loan
- The user has one overdue book
  - 2b The system denies the loan

Library Example:

- Set of initial CRC cards
- **Librarian:** the object in the system that fulfills User requests to check out, check in, and search for library materials
- **Borrower:** the set of objects that represent Users who borrow items from the library
- **Book:** The set of objects that represent items to be borrowed from the library



LIBRARIAN

CHECK OUT BOOK | BORROWER, BOOK

BOOK

KNOW IF OVERDUE
KNOW DUE DATE
CHECK OUT
CALCULATE DUE DATE
KNOW BORROWER

DATE
BORROWER

DATE

COMPARE DATES | DATE

BORROWER

CAN BORROW
KNOW SET OF BOOKS

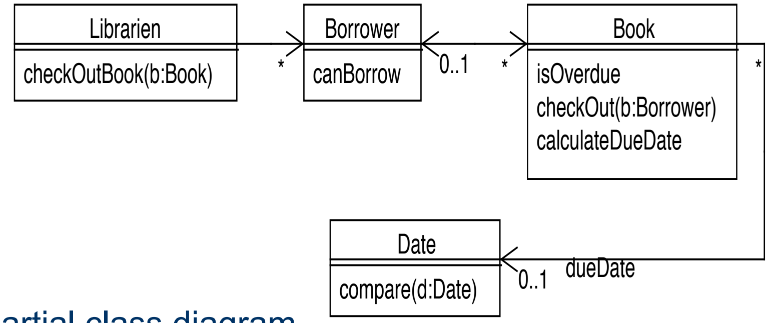
BOOK

OOA with CRC cards

Next Steps:

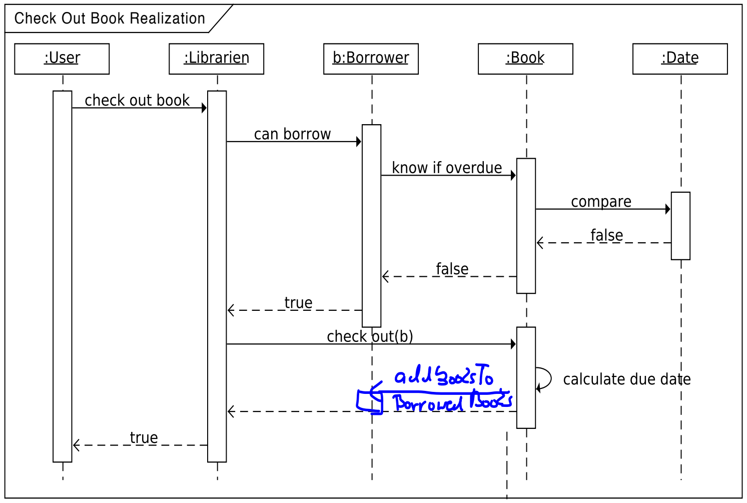
- Review the result
  - Group cards
  - Check cards
  - Refactor
- Transfer the result
  - UML class diagram
    UML interaction diagrams

Class Diagram



Partial class diagram

Sequence Diagram Diagram



Example: Avoid God Classes

Compute the price of an order

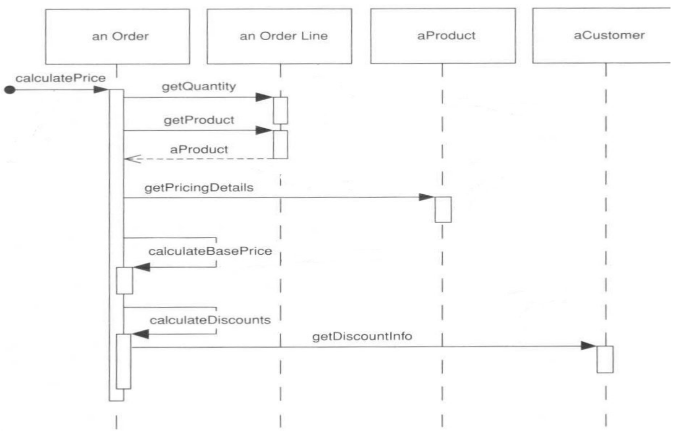
Order	
calculate price knows order lines knwos custommer	Order Line Customer

Customer	
knows name knows discount info	

OrderLine	
knows quantity knows product	Prodcut

Product	
knows price knows name	

Order does all the computation



Centralized control:

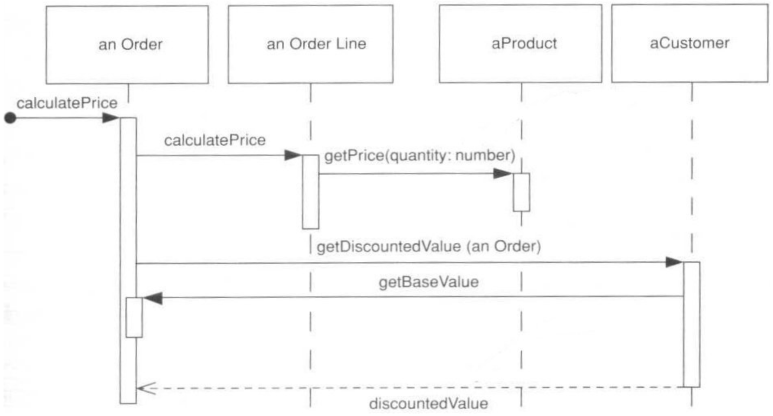
Order	
calcuate price calcuate base price calcuate discounts knows order line knows customer	Order Line Product Customer

Customer	
knows name knows discount info	

OrderLine	
knows quantity knows product	Prodcut

Product	
knows price knows name	

Let Order delegate computation



Distributed Control

Order	
calcuate price	Order Line
knows base price	Customer
calcuate discounts	
knows order line	
knows customer	

Customer	
knows name	Order
knows discount info	
calculate discount for order	

OrderLine	
knows quantity	Prodcut
knows product	
calculate price	

Product	
knows price	
knows name	
get price for quantity	