Software Engineering I (02161)

Week 5

Assoc. Prof. Hubert Baumeister

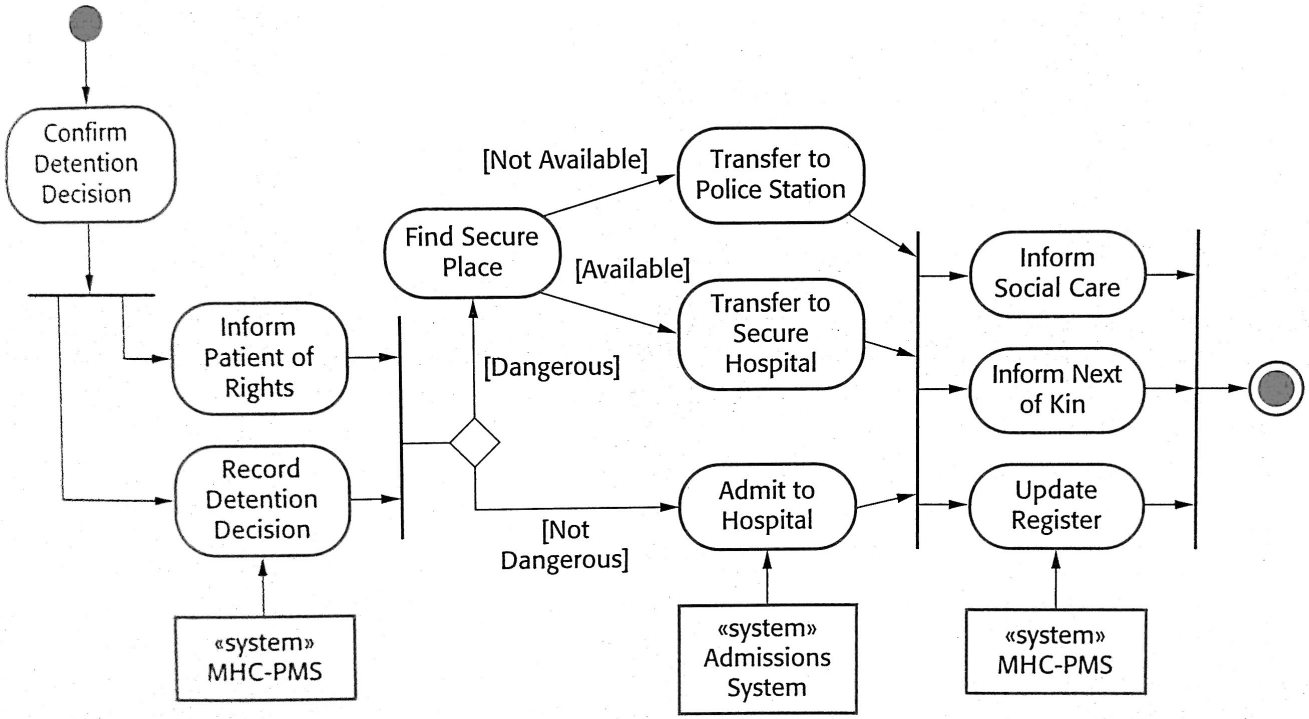
Informatics and Mathematical Modelling
Technical University of Denmark

Spring 2011

Recap: Class Diagrams

- Class diagram: Visualize OO programs (i.e. based on OO programming languages)
  - → However, have more abstract language
- Classes: combines data and methods related to a common aspect (e.g. Person, Address, Company, . . .)
- Generalization between classes (corresponds to inheritance)
- Association between classes
  - Unidirectonal
    - Associations vs. attributes
  - Multiplicities and how to implement them: 0..1, 1, *
  - Bi-directional
  - Qualifed assocations: Corresponds to the use of maps or dictionaries
  - Aggregation and Composition

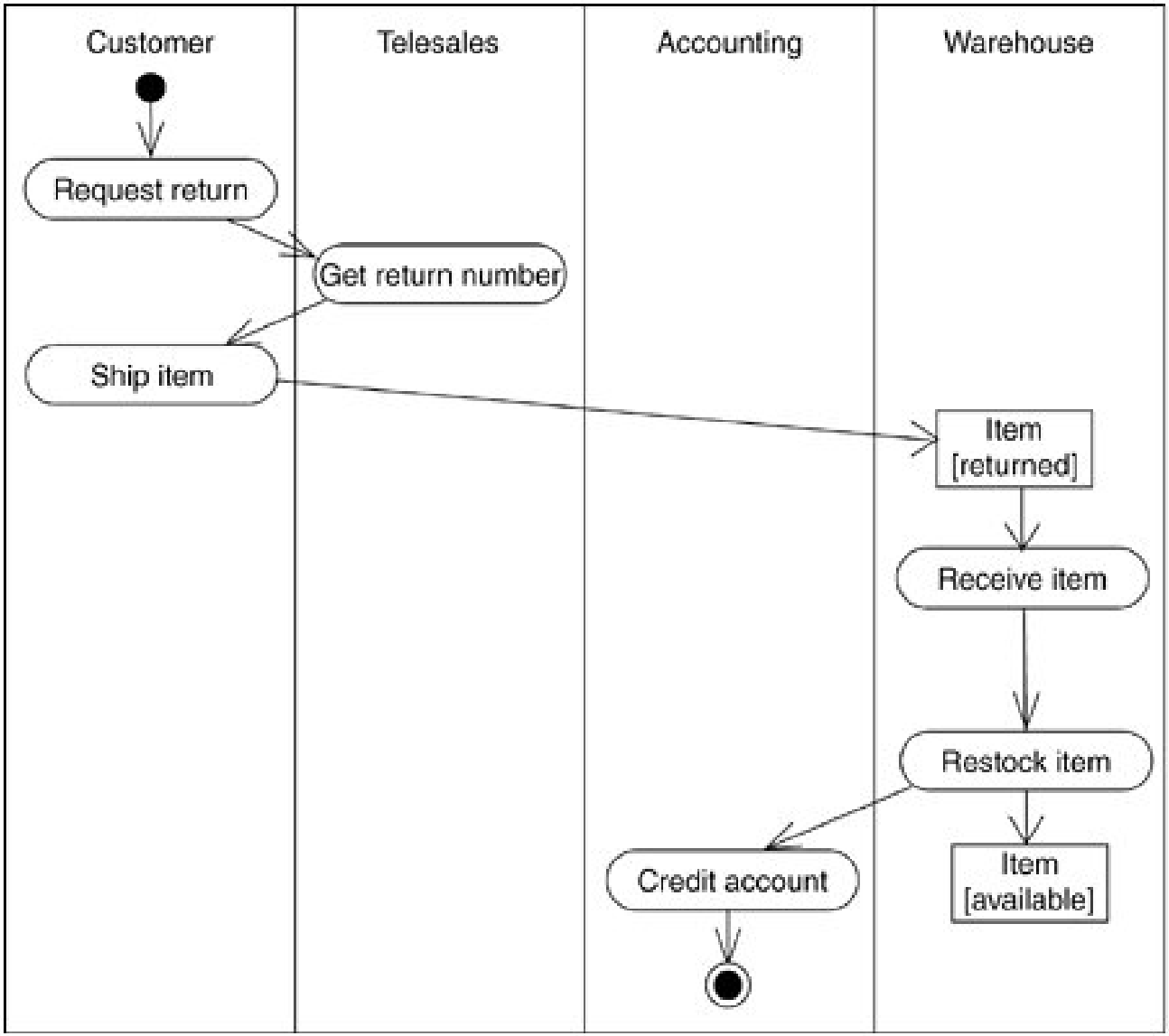
Activity Diagram: Business Processes



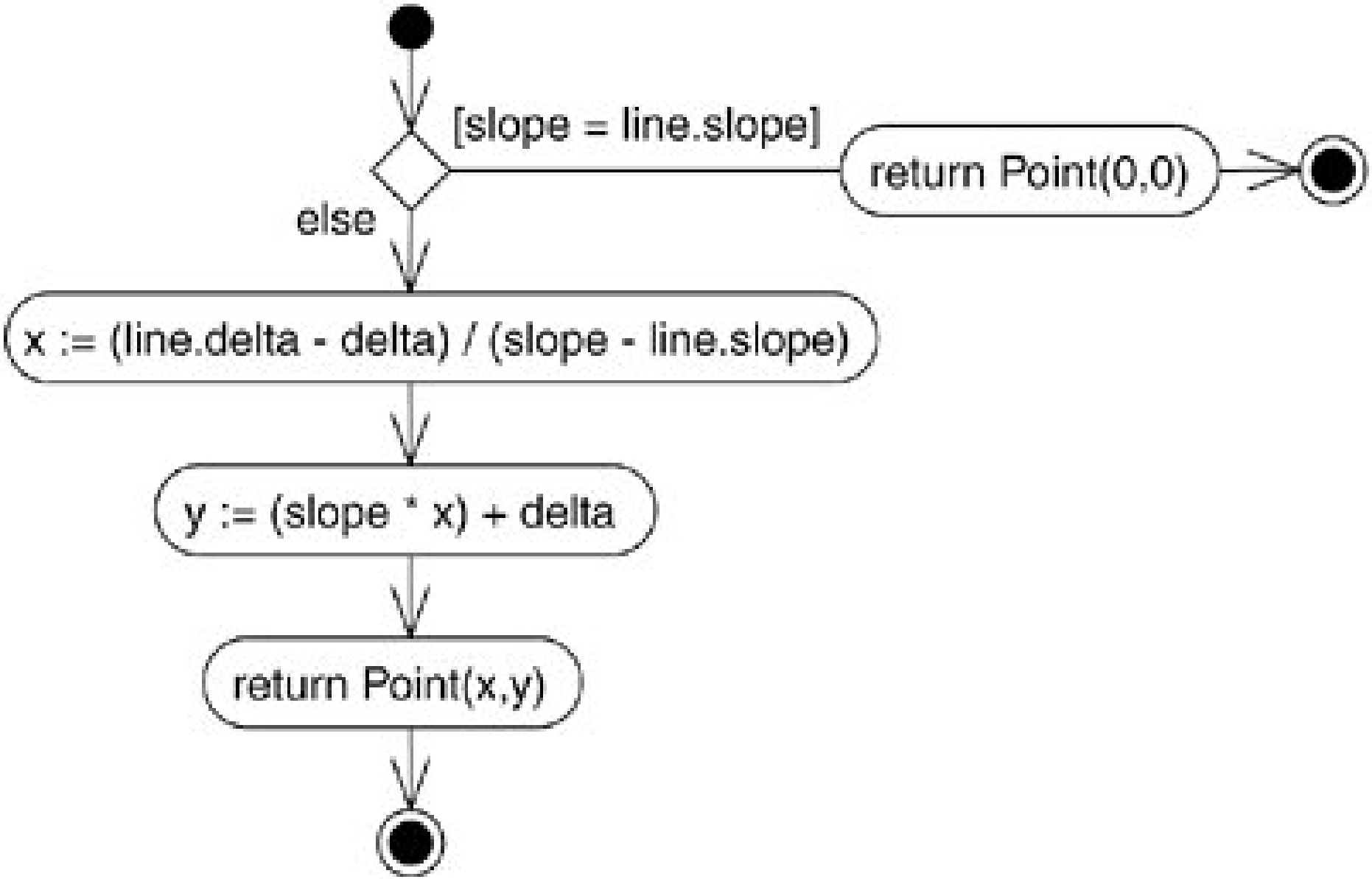
- Describe the context of the system
- Helps finding the requirements of a system
  - modelling business processes leads to suggestions for possible systems and ways how to interact with them
  - Software systems need to fit in into existing business processes

Ian Sommerville, Software Engineering – 9, 2010

Activity Diagram Example Workflow



Activity Diagram Example Operation

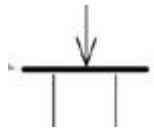
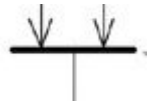
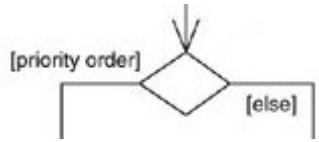


UML Activity Diagrams

- Focus is on control flow and data flow
- Good for showing parallel/concurrent control flow
- Purpose
  - Model business processes
  - Model workflows
  - Model single operations
- Literature: UML Distilled by Martin Fowler

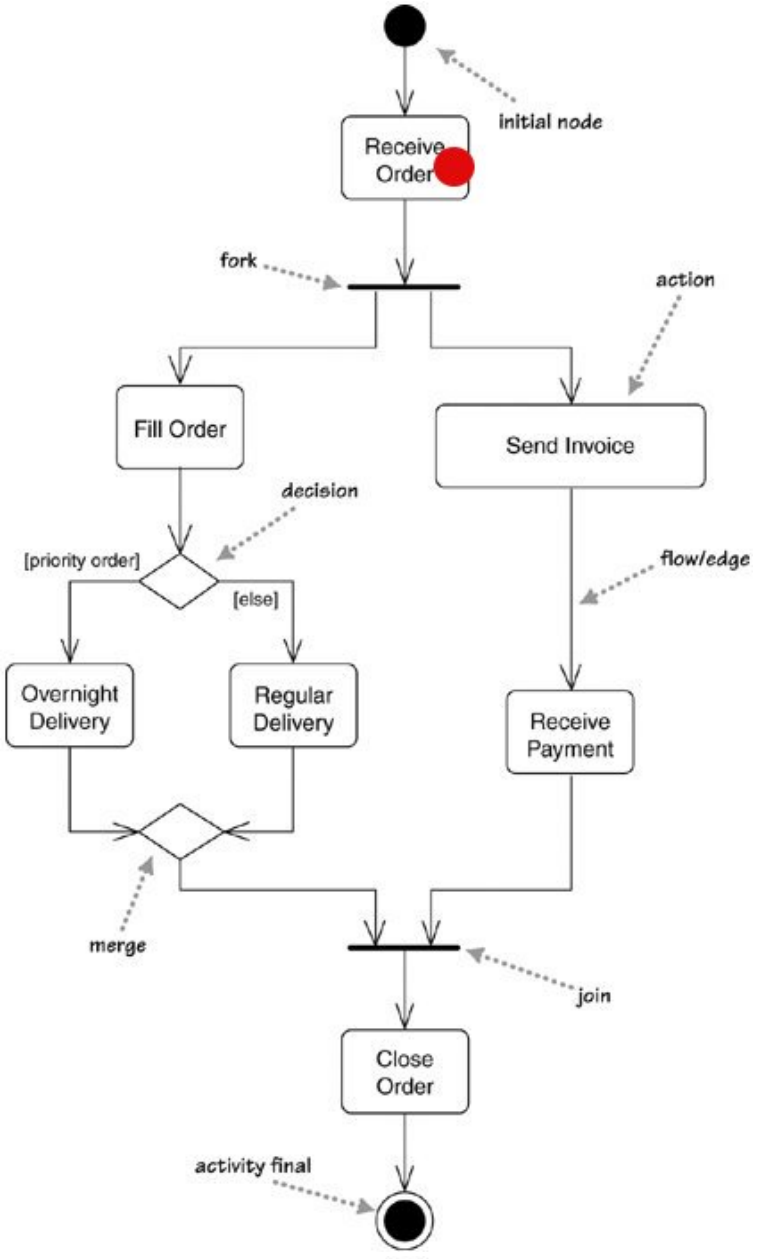
Activity Diagram Concepts

- Actions 

  - Are atomic
  - E.g Sending a message, doing some computation, raising an exception, . . .
    - UML has approx. 45 Action types
- Concurrency
  - Fork: Creates concurrent flows 
    - Can be true concurrency
    - Can be interleaving
  - Join: Synchronisation of concurrent activities 
    - Wait for all concurrent activities to finish (based on token semantics)
- Decisions 

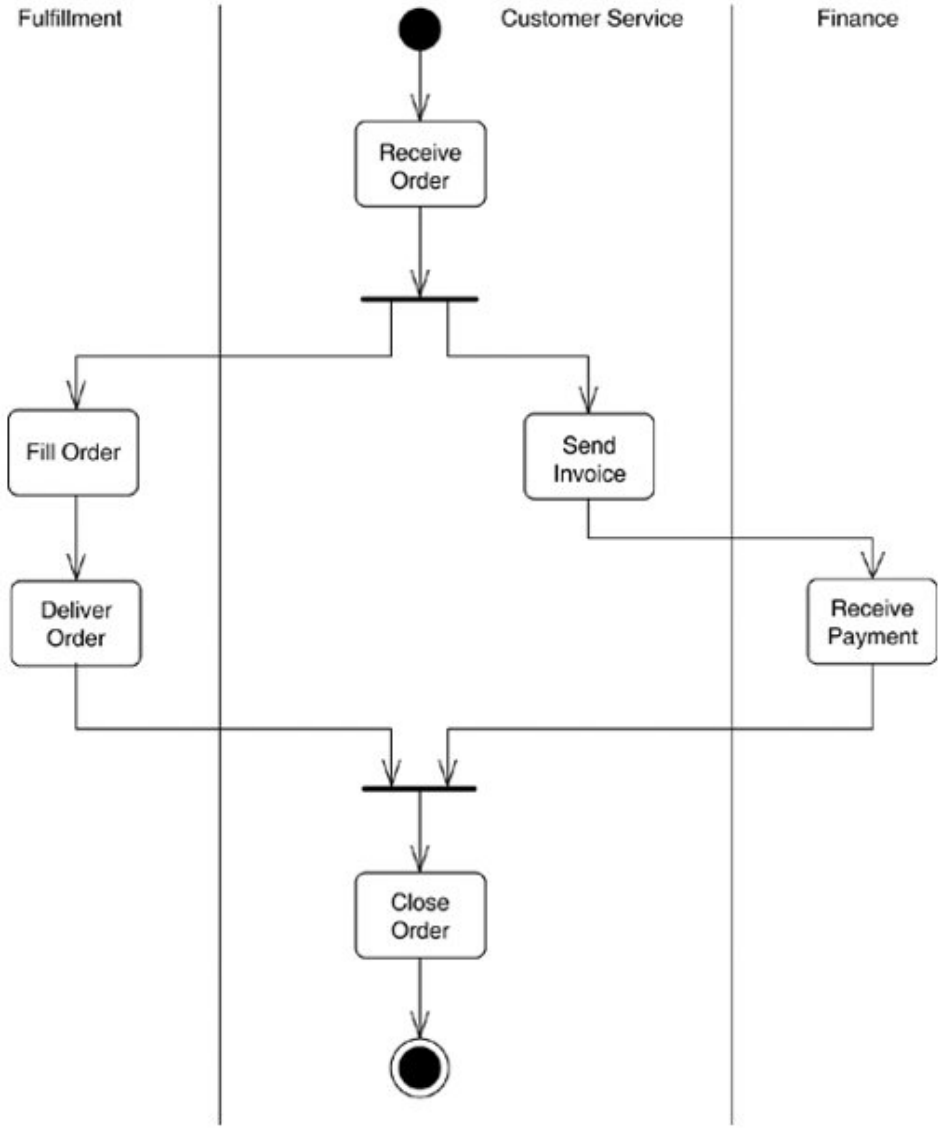
  - Notation: Diamond with conditions on outgoing transitions
  - `else` denotes the transition to take if no other condition is satisfied

Activity Diagrams Execution

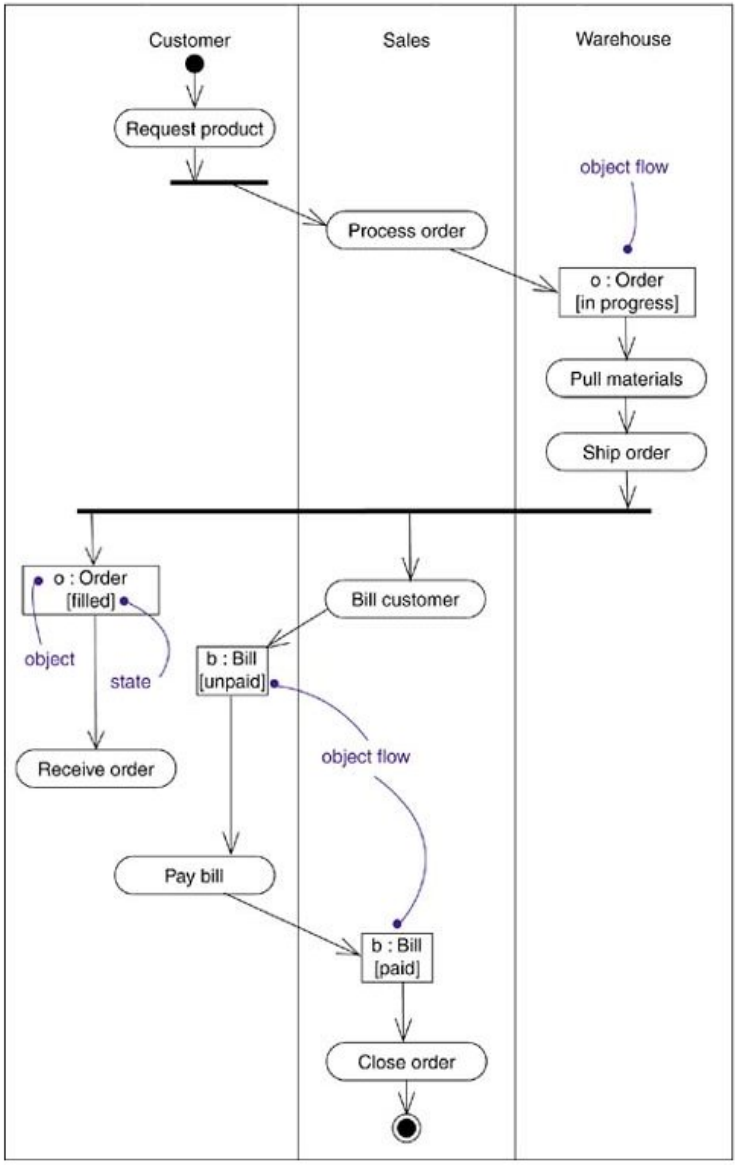


Swimlanes / Partitions

- Swimlanes show who is performing an activity

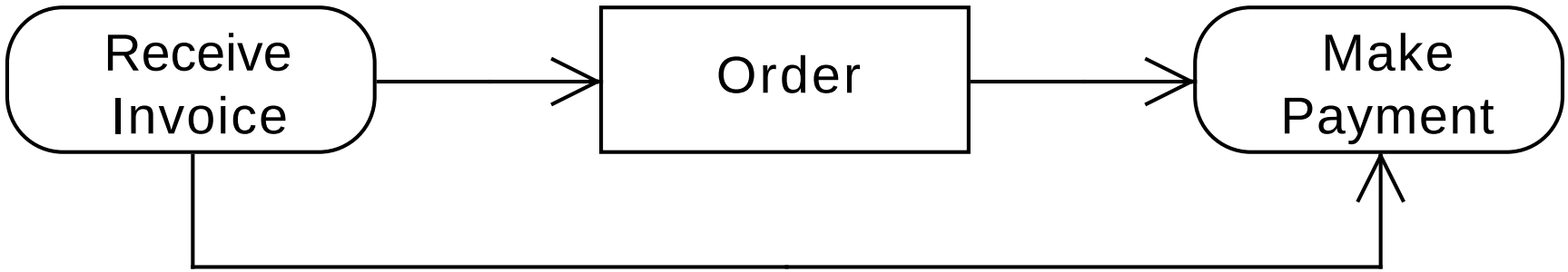


Objectflow example



Data flow and Control flow

- Data flow and control flow are shown:



- Control flow can be omitted if implied by the data flow:

Use of Activity Diagrams

- Emphasise on concurrent/parallel execution
- Requirements phase
  - To model business processes / workflows to be automated
- Design phase
  - Show the semantics of one operation
    - Close to a graphic programming language
- An activity diagram only shows one perspective of the dynamic aspect of the system
  - Use several activity diagrams and don't put everything into one diagram

Activities in Software Developement

- Understand and document what kind of the software the customer wants
  - → Requirements Analysis
- Determine how the software is to be built
  - → Design
- Build the software
  - → Implementation
- Validate that the software solves the customers problem
  - → Testing
  - → Verification
  - → Evaluation: e.g. User friendlieness

From Requirements to Design: Solution

Design process

1. The terms in the glossary give first candidates for classes, attributes, and operations
2. Take one use cases
   a. Take one main or alternative scneario
      i. Realize that scenario by adding new classes, attributes, associations, and operations so that you design can execute that scenario
      (ii. implement the design) (in case of an agile software development process)
   b. Repeat step a with the other scenarios of the use case
3. Repeat step 2 with the other use cases

- Techniques that can be used
  - Grammatical analysis of the text of the scenario
    → nouns are candidate for classes and attributes; verbs are candidates for operations, and adjectives are candidates for attributes
  - CRC cards (= Class Responsibility Collaboration cards)

Introduction CRC Cards

- CRC cards were developed by Ward Cunningham in the late 80's
- Can be used for different purposes
  - Analyse a problem domain
  - Discover object-oriented designs
    - Learn to think objects

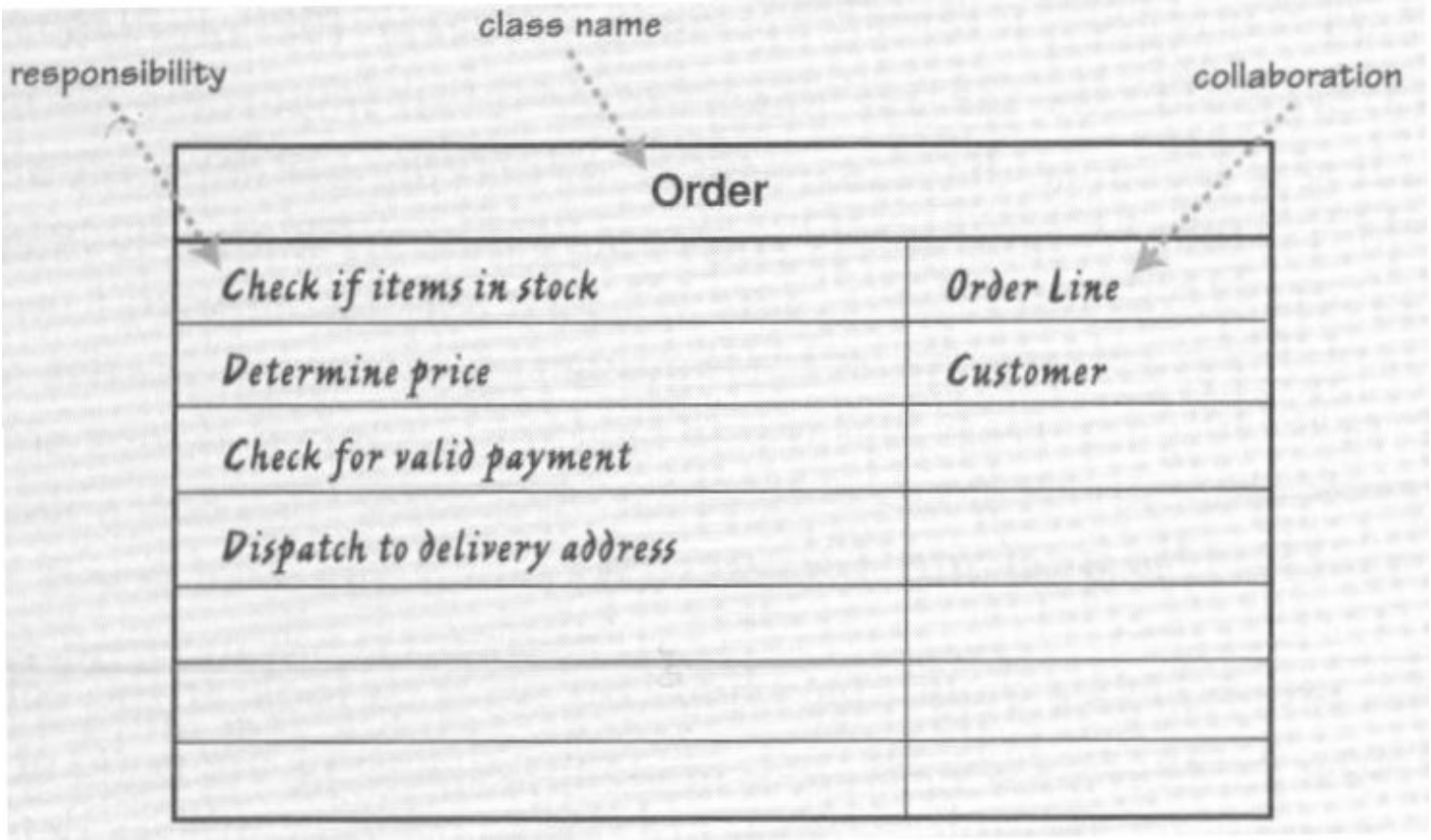
  → Objects
    - have structure and behaviour

    → both need to be considered at the same time
- Literature
  - `http://c2.com/doc/oopsla89/paper.html`
  - Martin Fowler: UML Destilled pages 62—63

CRC Card

- Class
  - Can be an object of a certain type
  - Can be the class of an object
  - Can be a component of a system
  - Index cards are used to represent classes (one for each class) (I use A6 paper instead of index cards)
- Responsibilities
  - Corresponds roughly to operations and attributes
  - Somewhat larger in scope than operations
  - ”do something”
  - ”know something”
- Collaborations
  - With whom needs this class to collaborate to realize its responsibilities

CRC Card Template



A larger example

- `http://c2.com/doc/crc/draw.html`

Process I

- Starting point
  - List of use-cases scenarios
    - Should be as concrete as possible
  - A group of up to 2–6 people
- Brainstorming
  - Initial set of Classes (just enough to get started)
  - Assign Classes to persons
- Execute Scenarios
  - Simulate how the computer would execute the scenario
  - Each object/class is represented by one person
  - This person is responsible for executing a given responsibility
    - This is done by calling the responsibilities of other objects/persons he collaborates with
  - objects/classes can be created
  - responsibilitites can be added
  - collaborations can be added

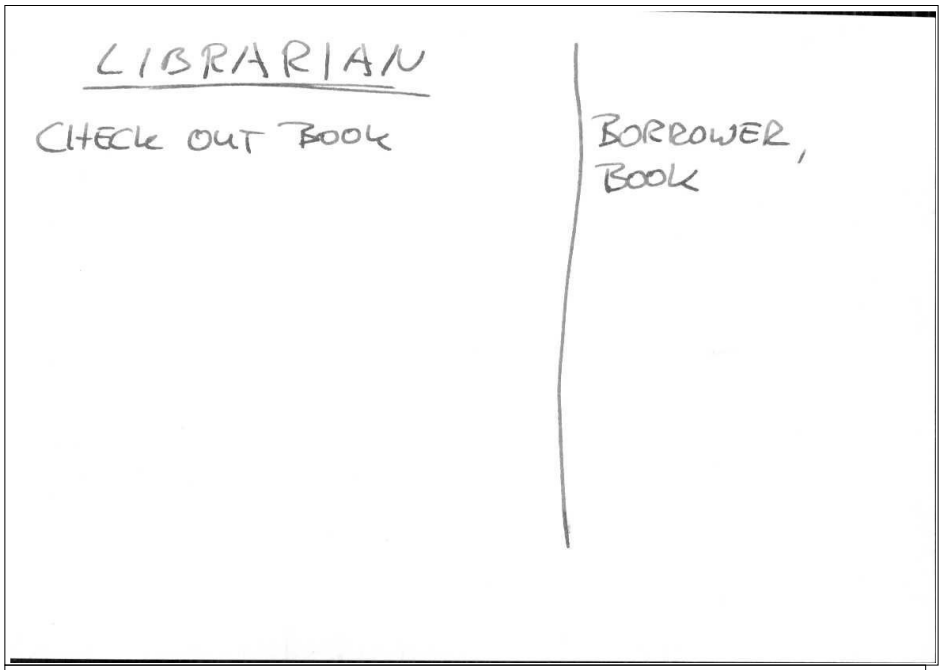
Library Example: Detailed Use Case Check Out Book

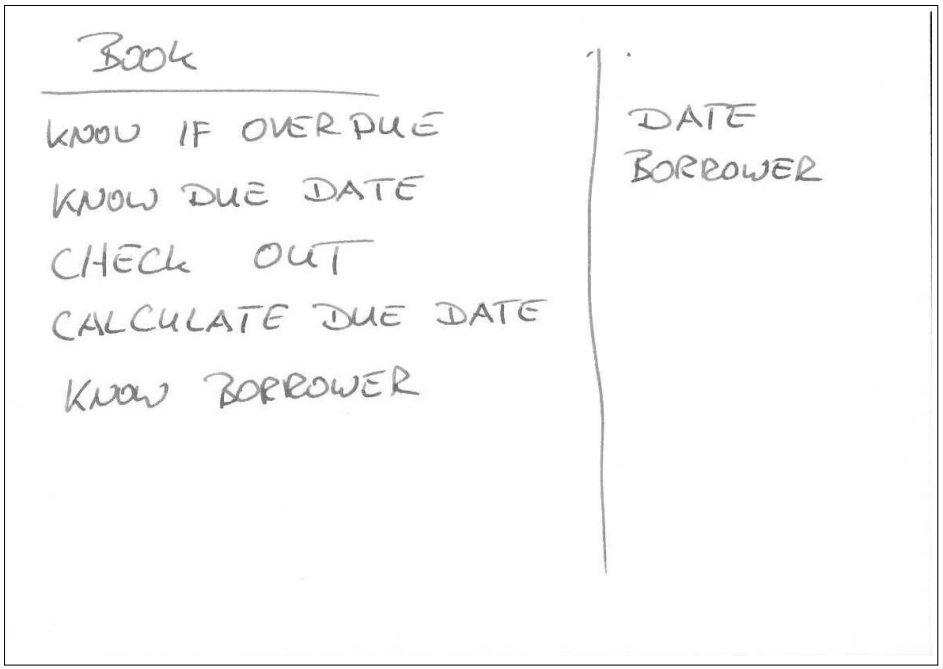
- Name: Check Out Book
- Description: The user checks out a book from the library
- Actor: User
- Main scenario:
  1. A user presents a book for check-out at the check-out counter
  2. The system registers the loan
- Alternative scenarios:
  - The user already has 10 books borrowed
    - 2a. The system denies the loan
  - The user has one overdue book
    - 2b. The system denies the loan

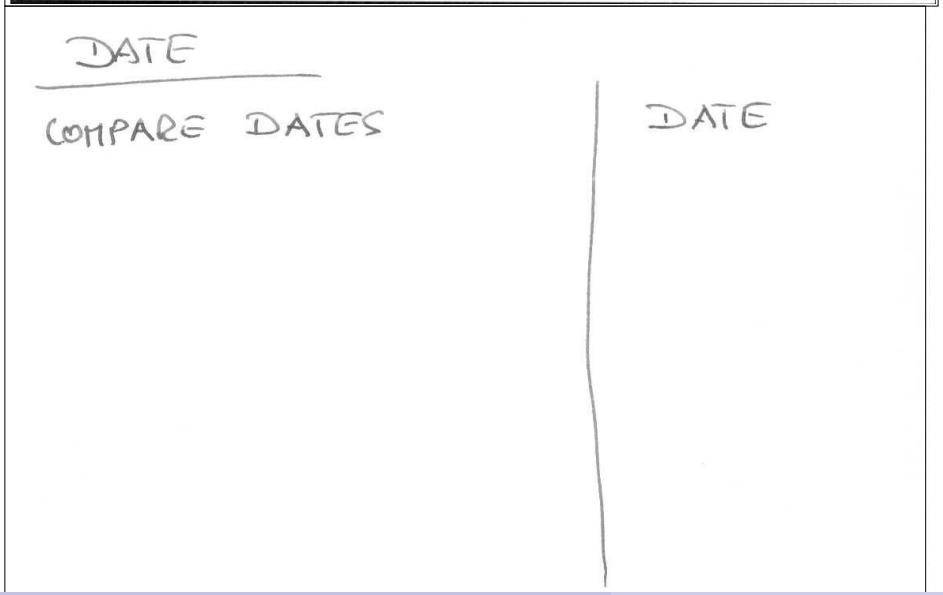
Example II

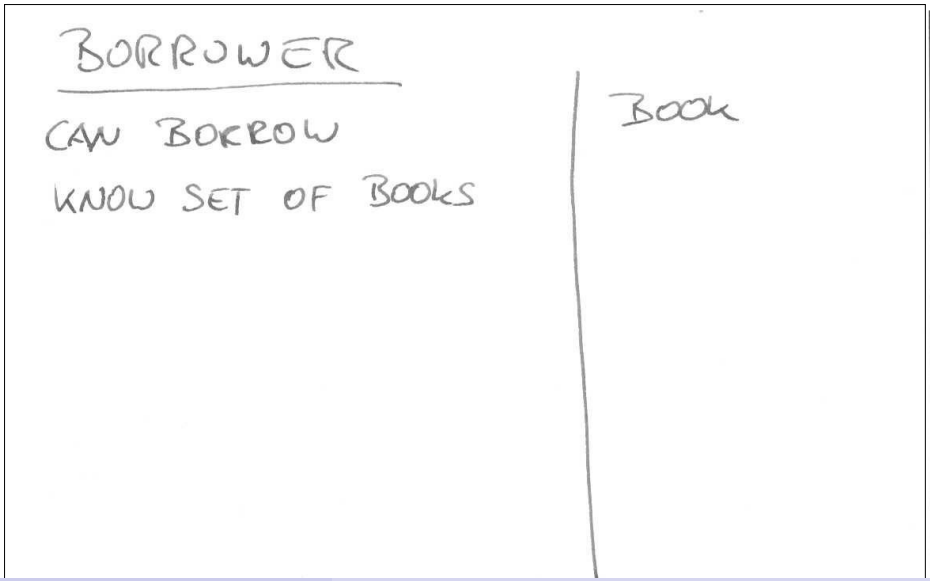
- Set of initial CRC cards
  - Librarien
    - The object in the system that fulfills User requests to check out, check in, and search for library materials
  - Borrower
    - The set of objects that represent Users who borrow items from the library
  - Book
    - The set of objects that represent items to be borrowed from the library
  - Use case Check out book main scenario
    - ”What happens when Barbara Stewart, who has no accrued fines and one outstanding book, not overdue, checks out a book entitled Effective C++ Strategies+?”

Library Example: All CRC cards





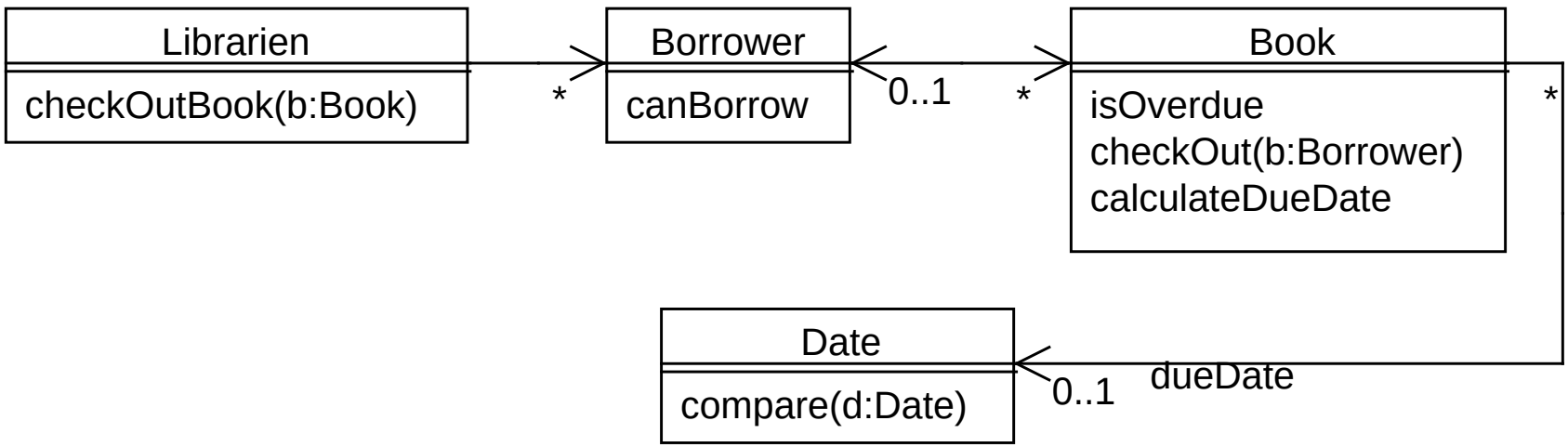




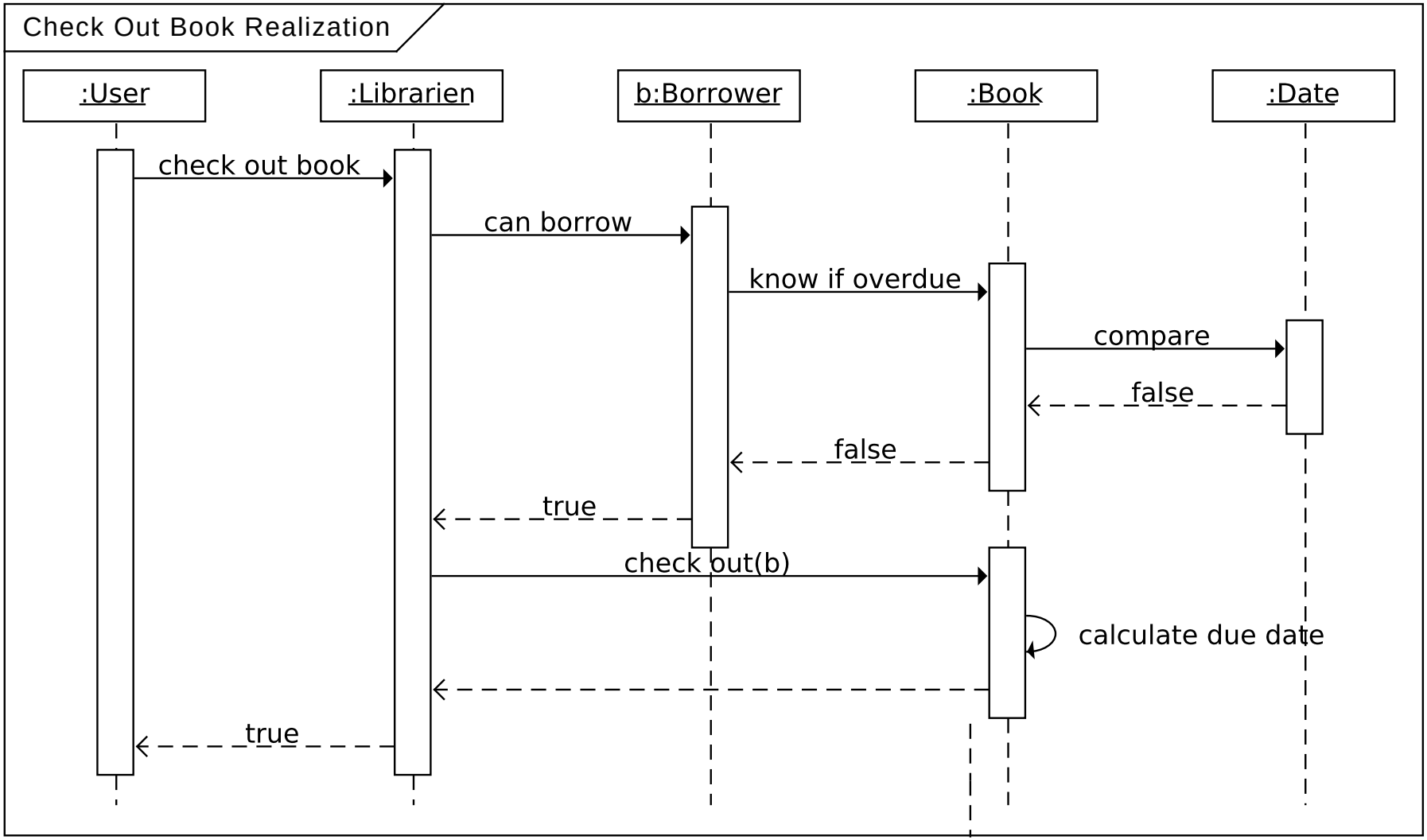
Process: Next Steps

- Review the result
  - Group cards
    - by collaborations
    - shows relationship between classes
  - Check responsibilities
  - Check correct representation of the domain
  - Refactor if needed
- Transfer the result
  - UML class diagrams
    - Responsibilities map to operations and attributes/associations
    - Collaborations map to associations and dependencies
  - The executed scenarios to UML interaction diagrams

Example: Class Diagram (so far)



Example: Sequence Diagram for Check-out book



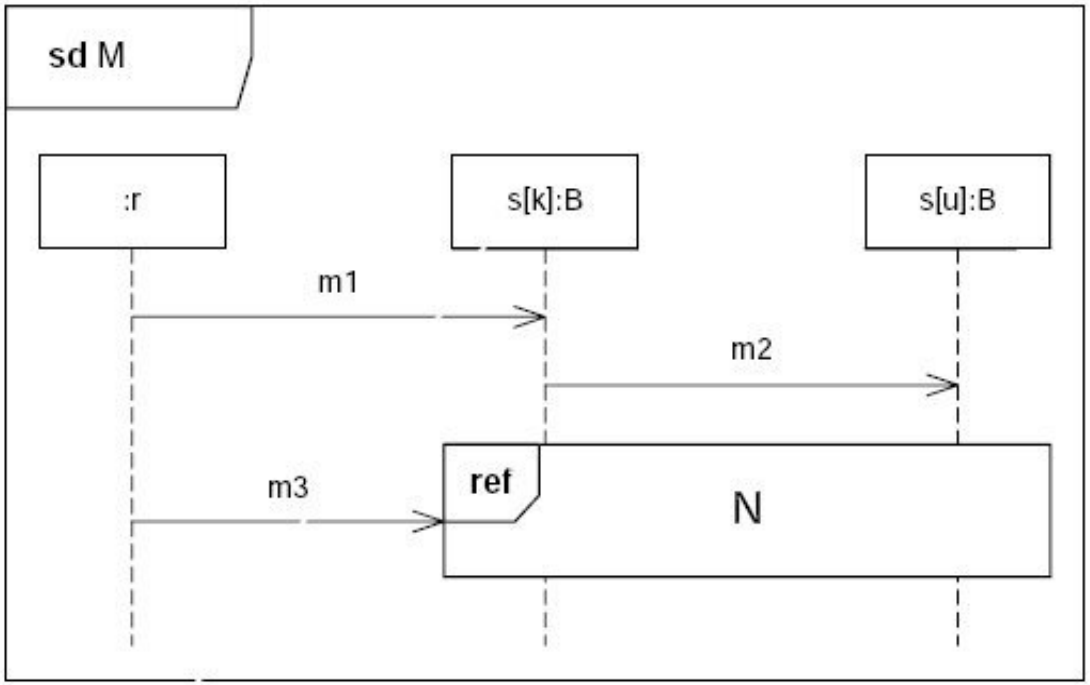
Summary

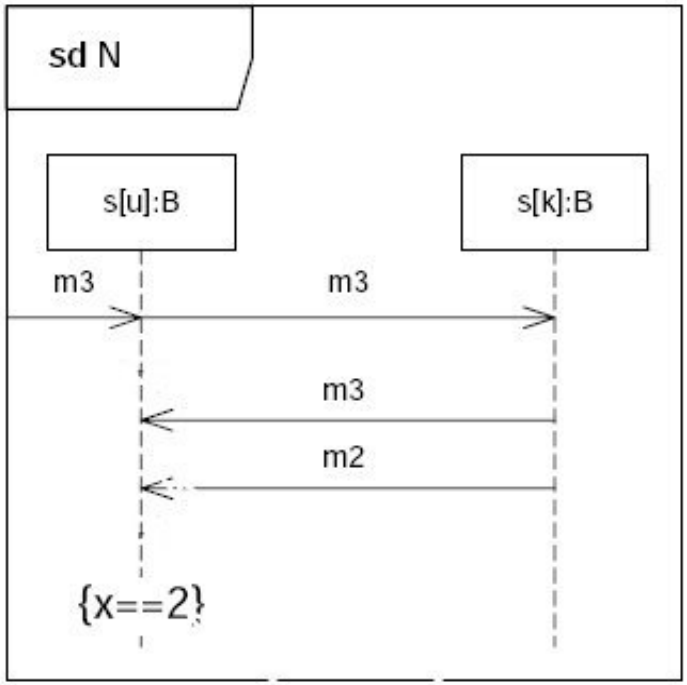
- Further scenarios give more detail
- The scenarios are now quite easy to implement
- CRC process can be repeated on a more detailed level, e.g., to design the database interaction, or user interface
- Helps one to think in objects (structure and behaviour)
- Humans playing objects help to get a better object-oriented design as it helps delegating responsibilities

Interaction Diagrams

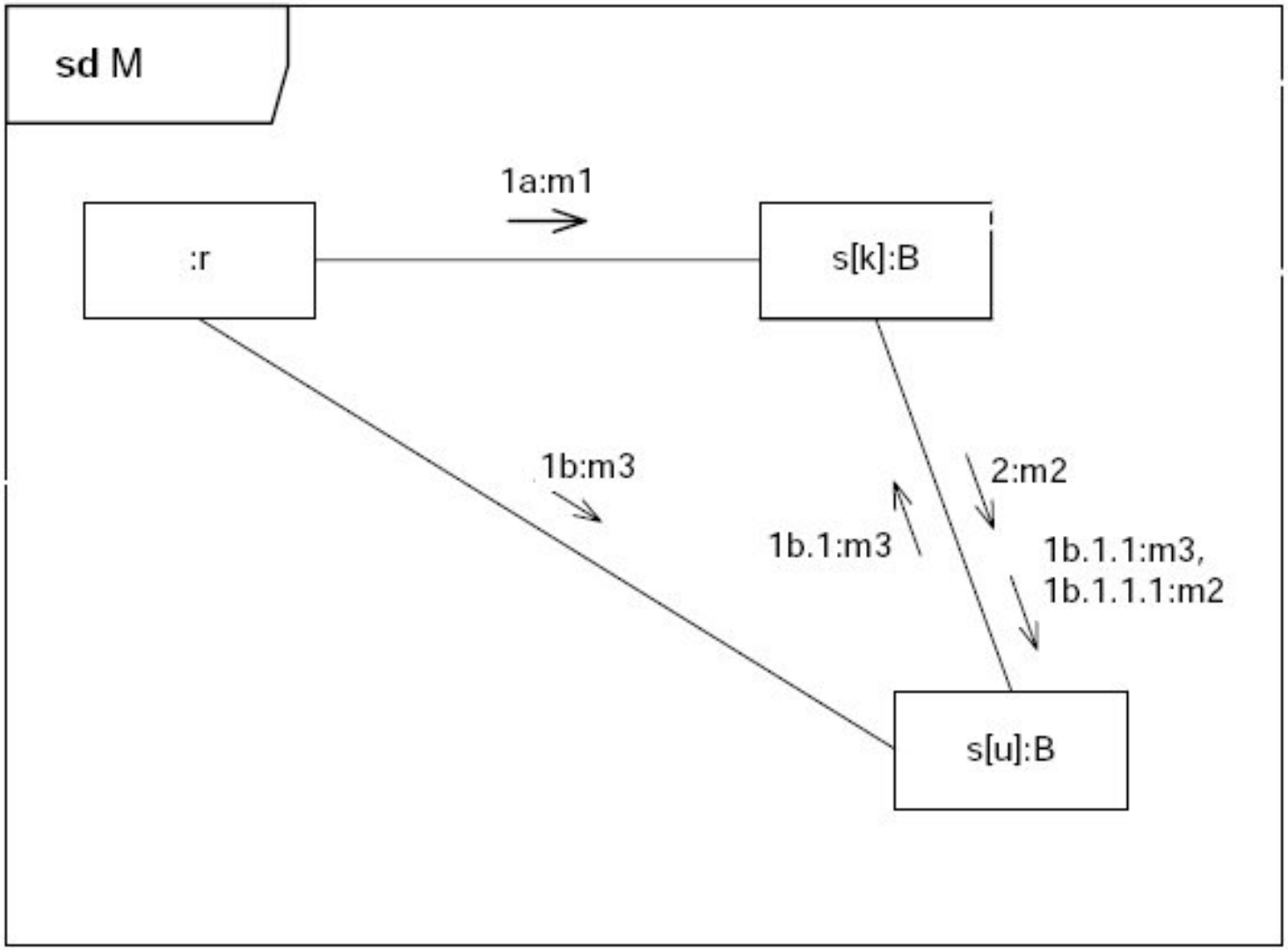
- Purpose
  - Describes how objects collaborate in some behaviour
- Types
  - Sequence Diagrams
    - 1990's: Message Sequence Charts (MSCs) used in TelCo-industry
  - Communication Diagrams
  - Timing Diagrams

Example Sequence Diagrams





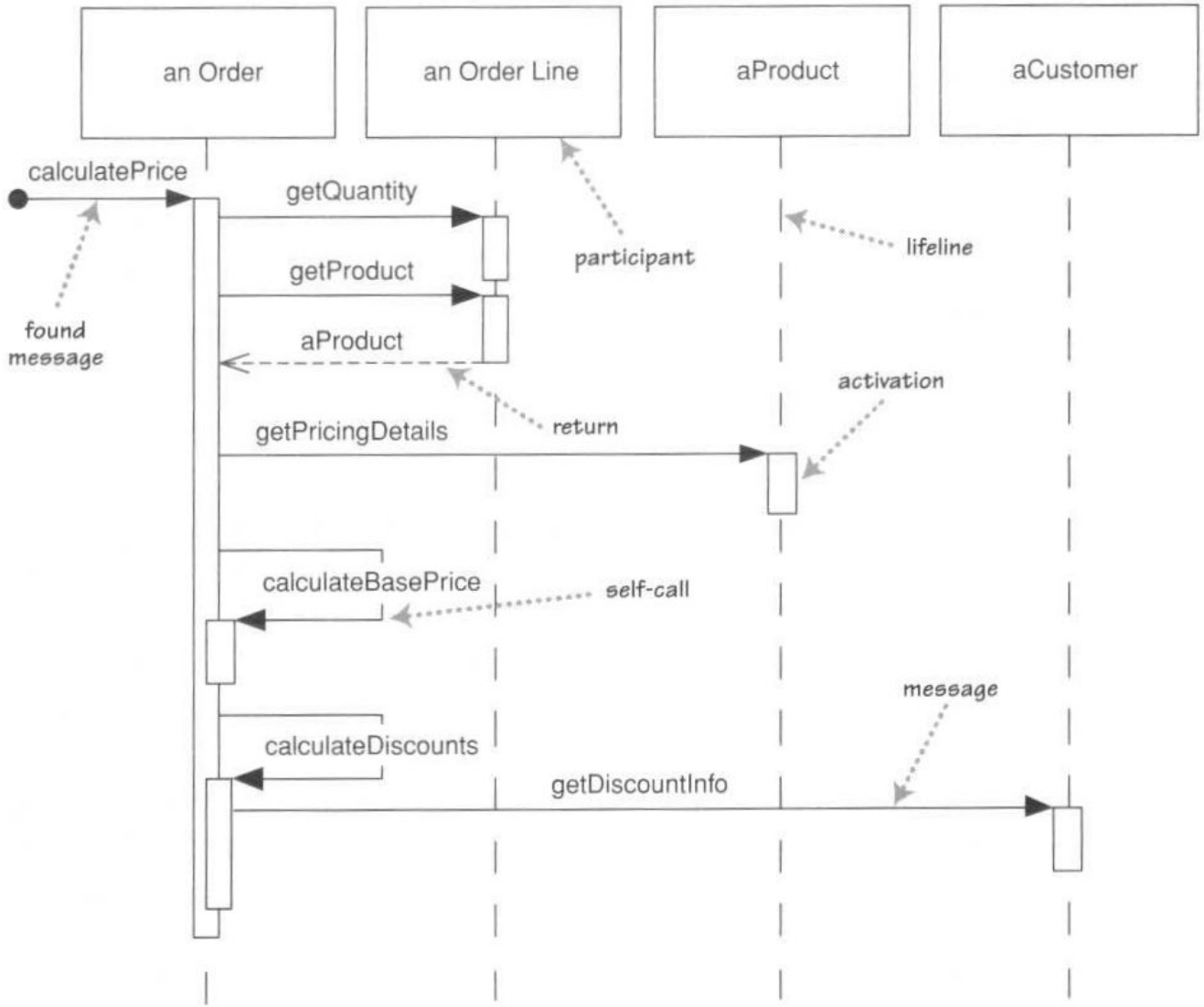
Example Communication Diagrams



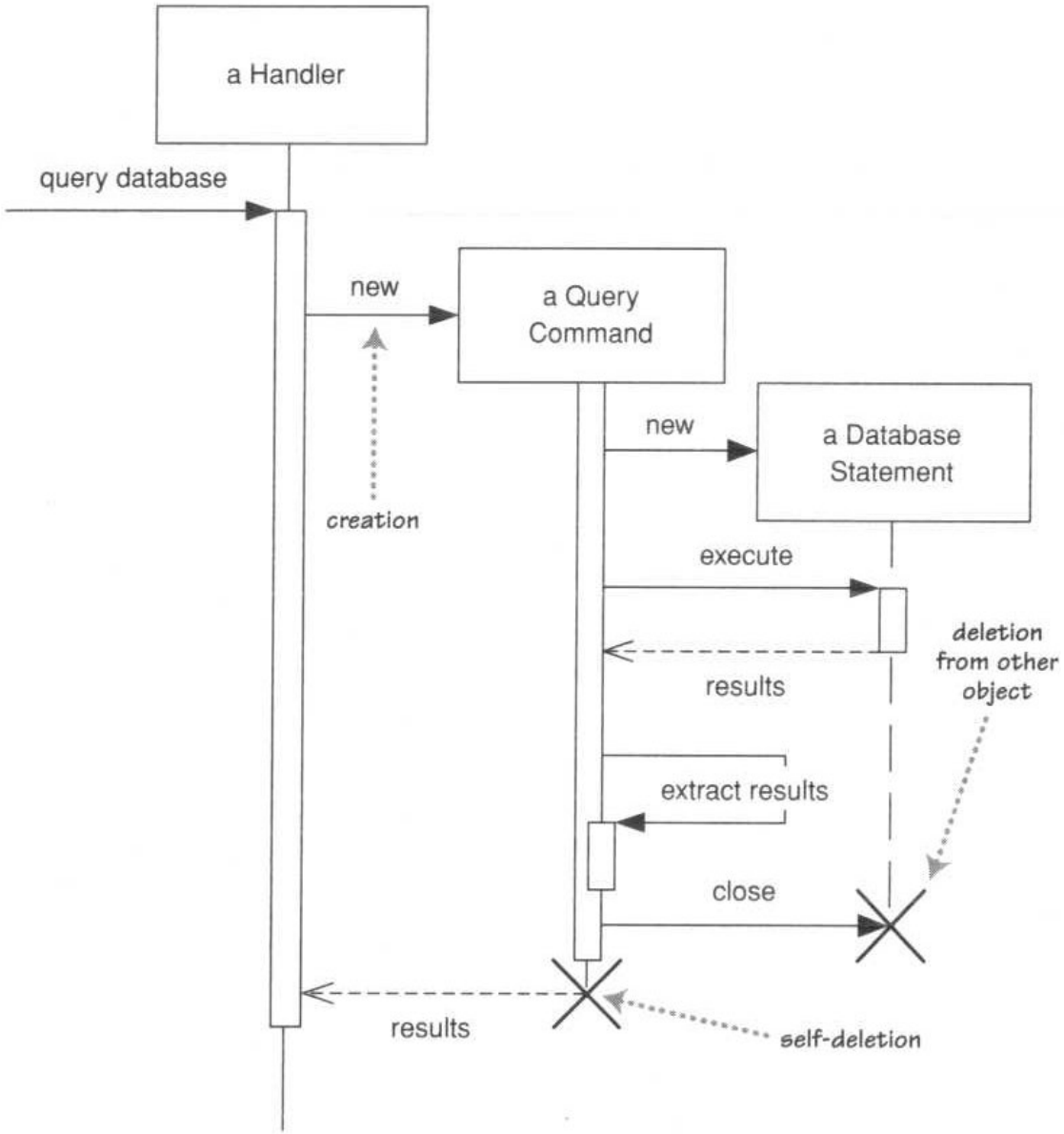
Use of Sequence Diagrams

- Usually a single scenario
  - But can be extended to include several interactions
  - → Executable UML / UML as a programming language
- Class/object interactions
  - Express message exchange between objects
  - Design message exchange
  - Get an overview of existing systems
- Use Case Scenarios
  - Illustrate a concrete scenario of a use case
  - Useful for design/documentation (Analysis, design, and reengineering)

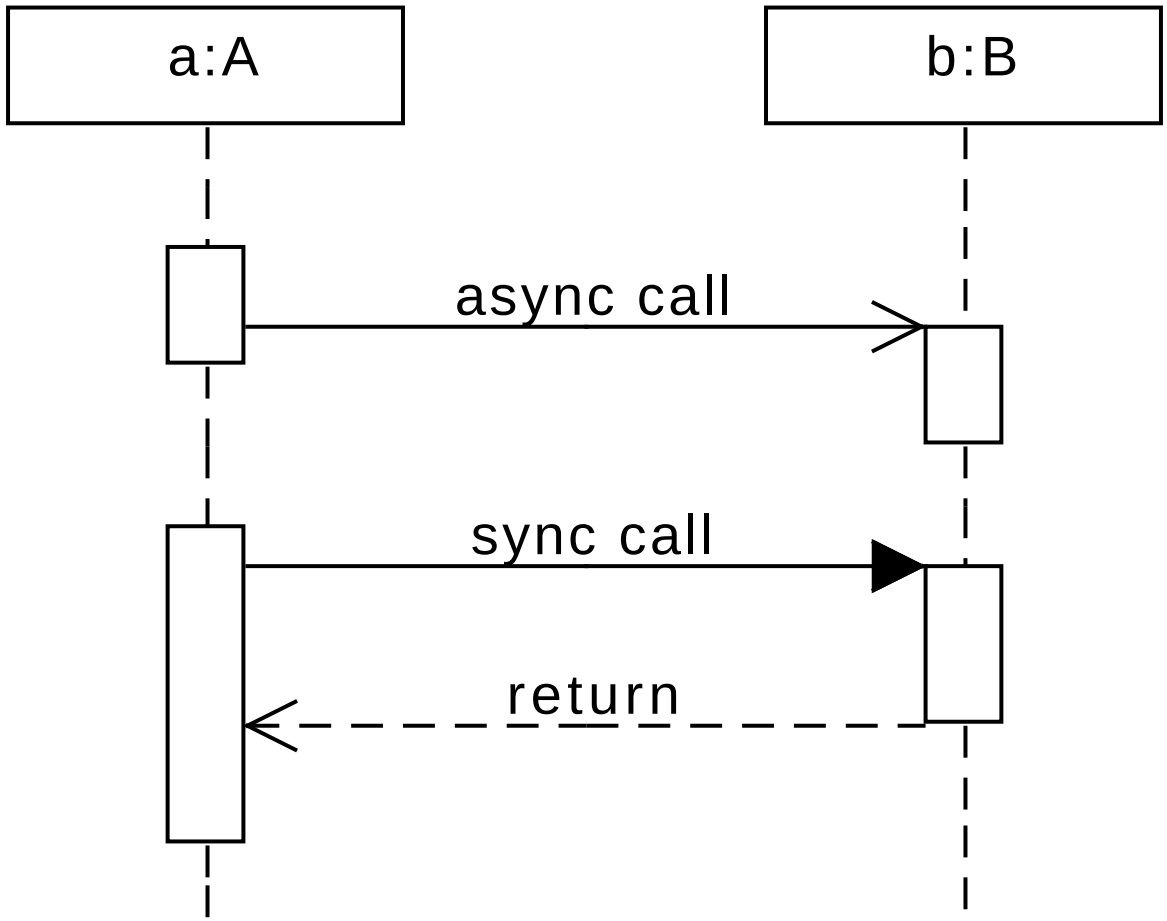
Sequence Diagram Concepts



Creation and deletion of participants



Arrow types

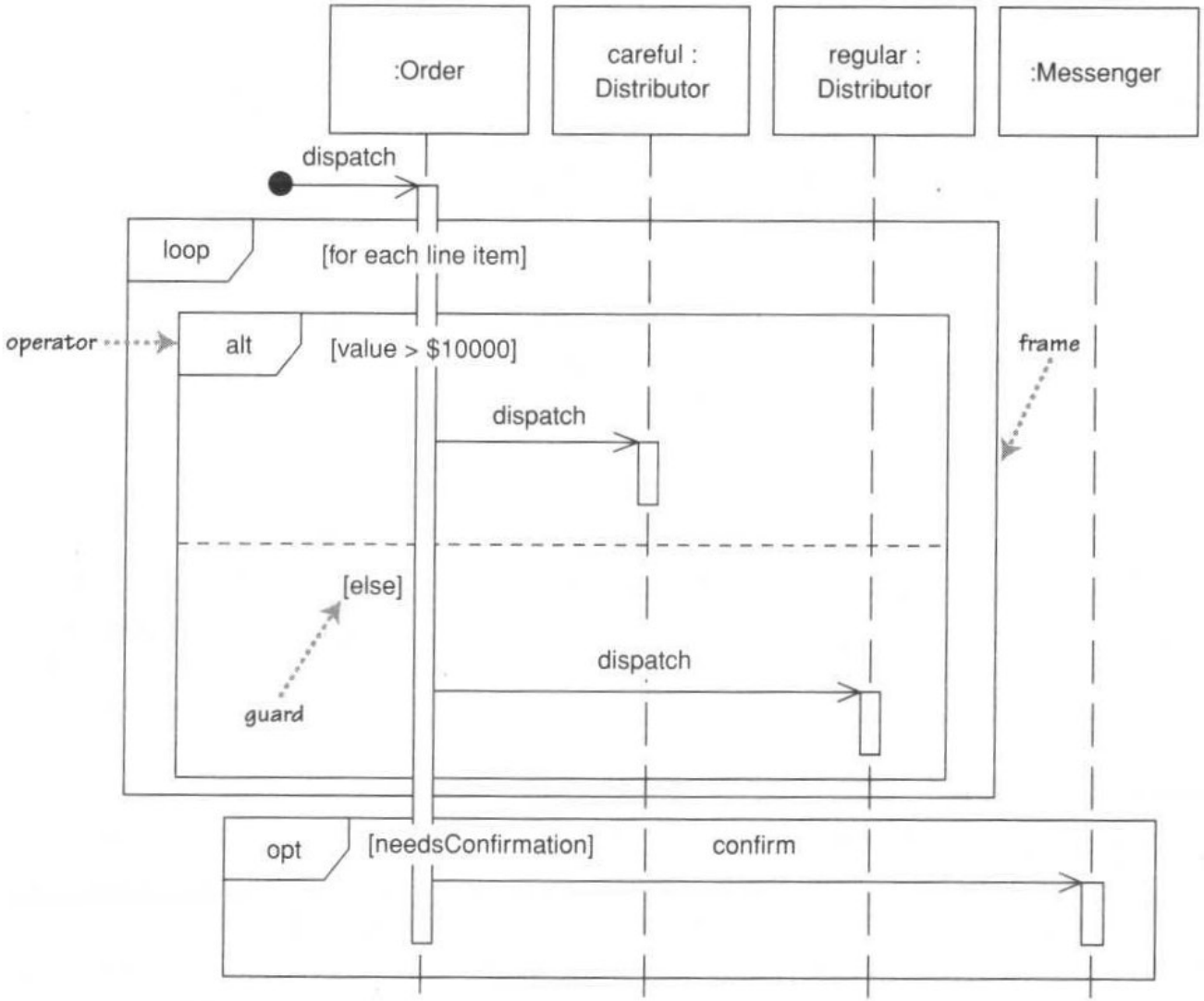


Interaction Frames Example

Realising an algorithm using a sequence diagram

```
public void dispatch() {
  for (LineItem lineItem : lineItems) {
    if (lineItem.getValue() > 10000) {
      careful.dispatch();
    } else {
      regular.dispatch();
    }
  }
  if (needsConfirmation()) {
    messenger.confirm();
  }
}
```

Realisation with Interaction Frames



Interaction Frame Operators I

Operator	Meaning
alt	Alternative multiple fragments; only the one whose condition is true will execute (Figure 4.4).
opt	Optional; the fragment executes only if the supplied condition is true. Equivalent to an alt with only one trace (Figure 4.4).
par	Parallel; each fragment is run in parallel.
loop	Loop; the fragment may execute multiple times, and the guard indicates the basis of iteration (Figure 4.4).
~~region~~ critical	Critical region; the fragment can have only one thread executing it at once.
neg	Negative; the fragment shows an invalid interaction.
ref	Reference; refers to an interaction defined on another diagram. The frame is drawn to cover the lifelines involved in the interaction. You can define parameters and a return value.
sd	Sequence diagram; used to surround an entire sequence diagram, if you wish.

When to use sequence diagrams

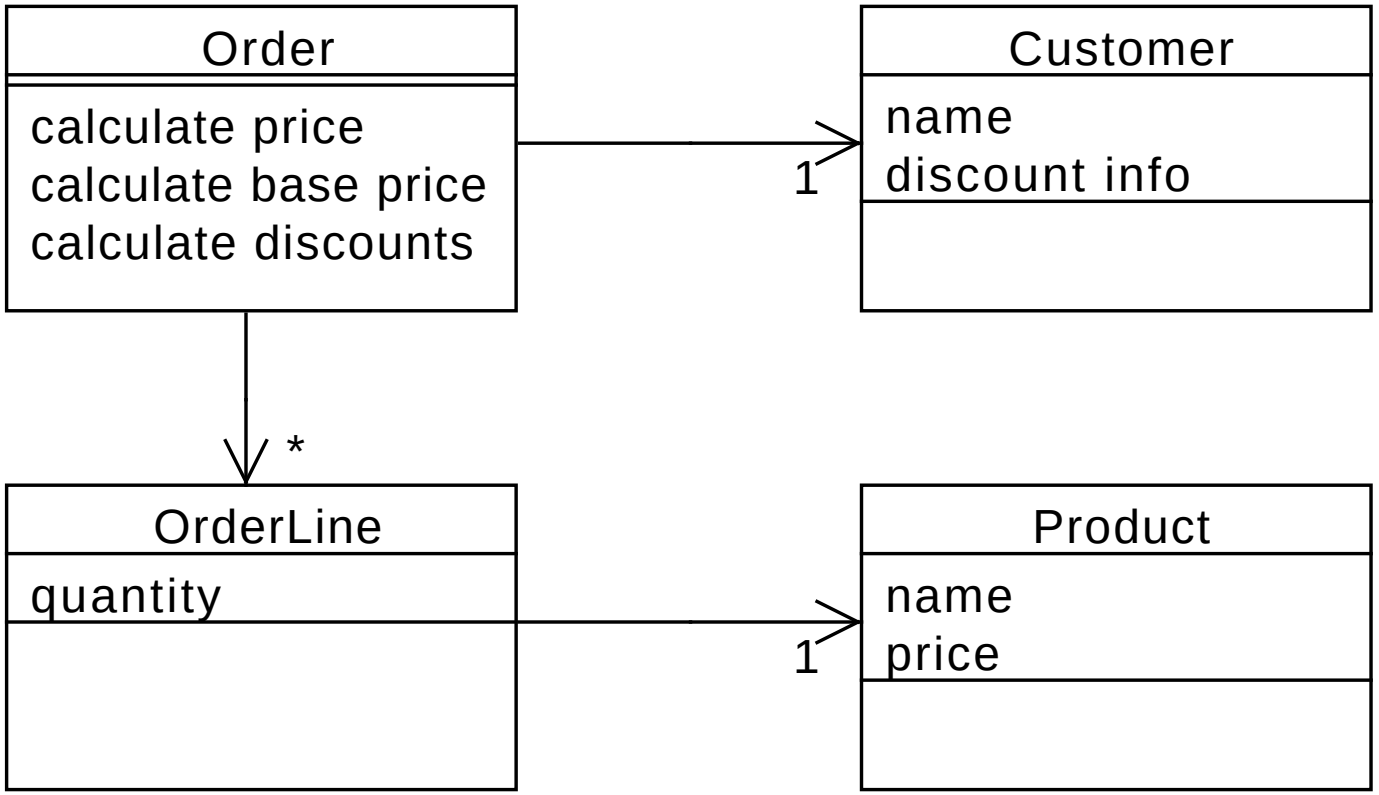
- Useful to visualize complex interactions between objects
  - Visualize the control flow in an object-oriented system
  - Reverse Engineering but also Forward Engineering
- Useful to describe Use Case scenarios
- Sequence diagrams are less useful for describing algorithms

  → Use state machines to show the behaviour of a single object across several scenarios

  → Use activity diagrams to show the behaviour of across many objects and threads

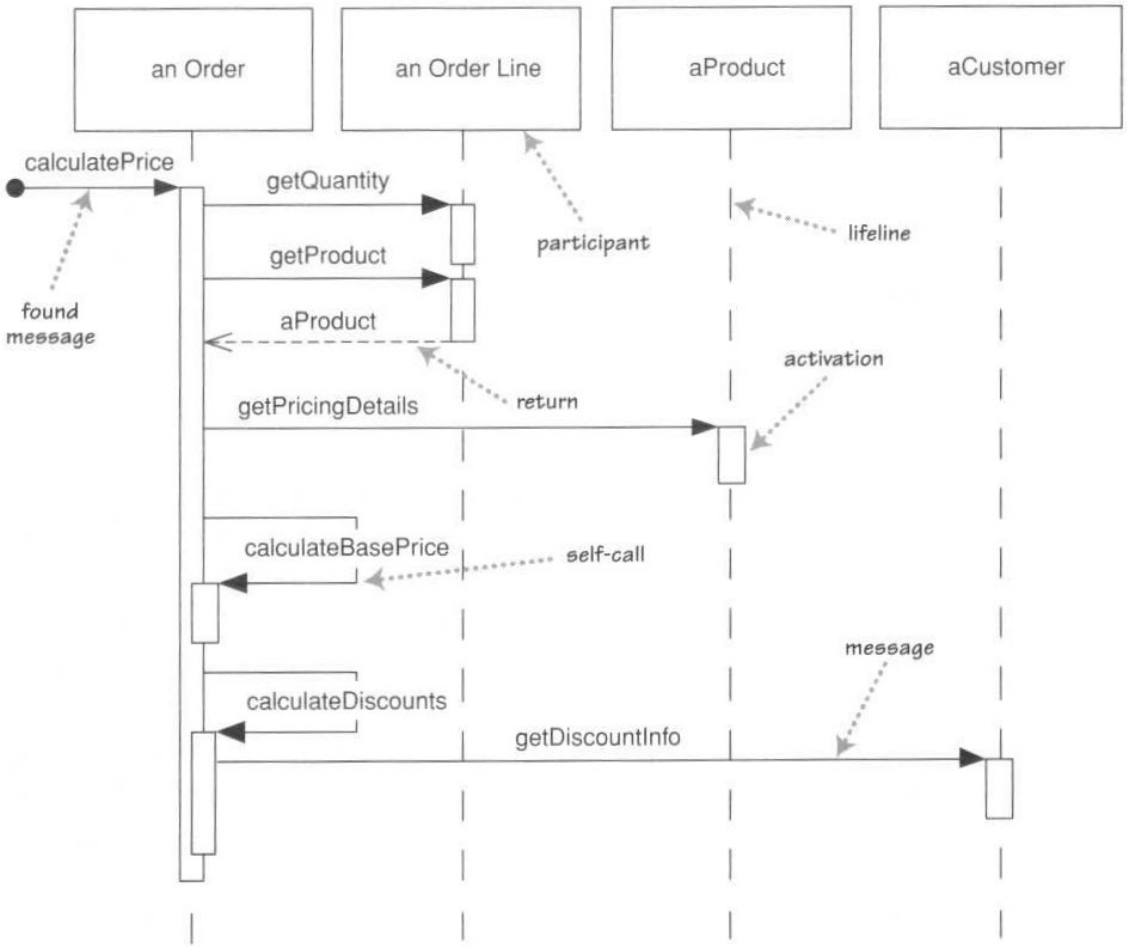
Computing the price of an order

- Task
  - Calculate the price of an order
  - Take into account if the customer has any discounts

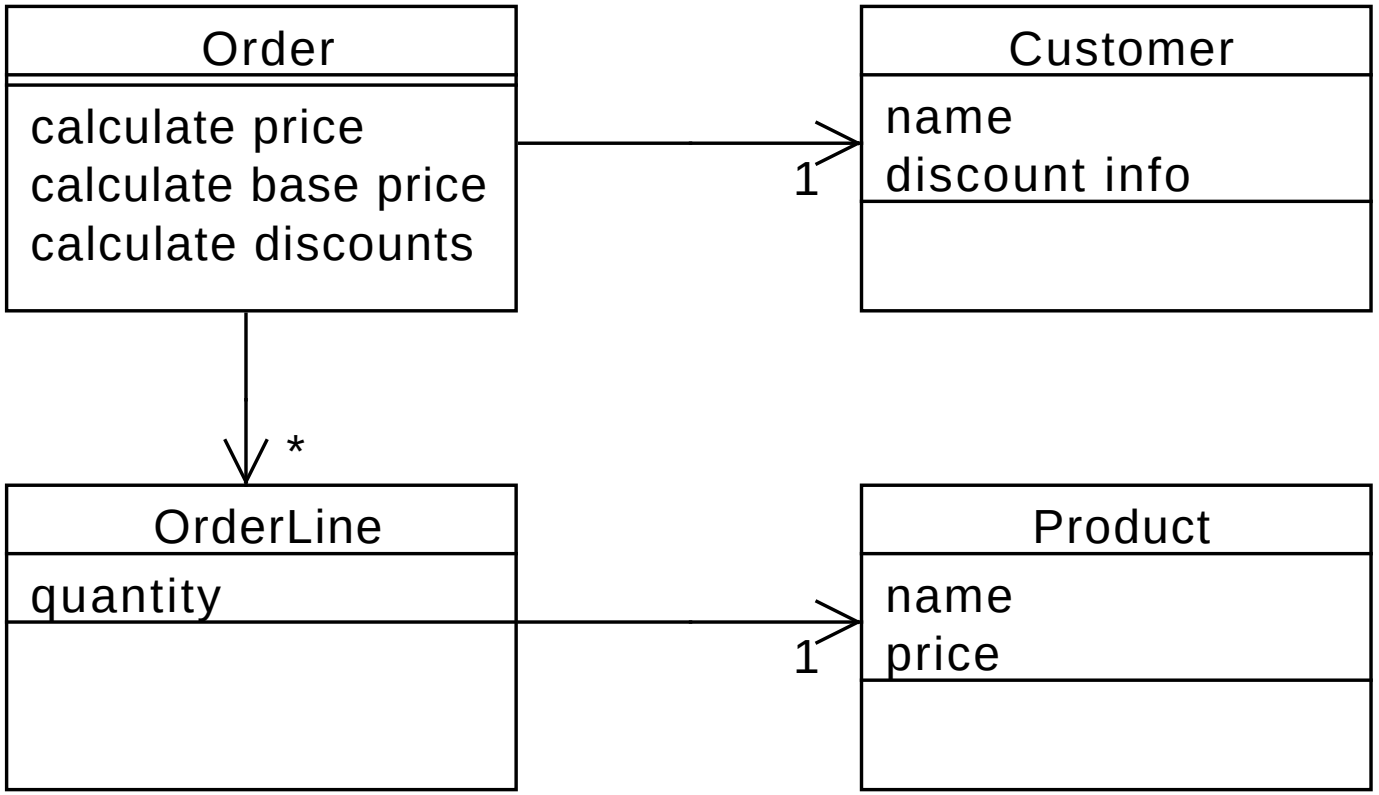


Computation is concentrated in one class (centralised control)

- The order computes the price by asking its collaborators about data



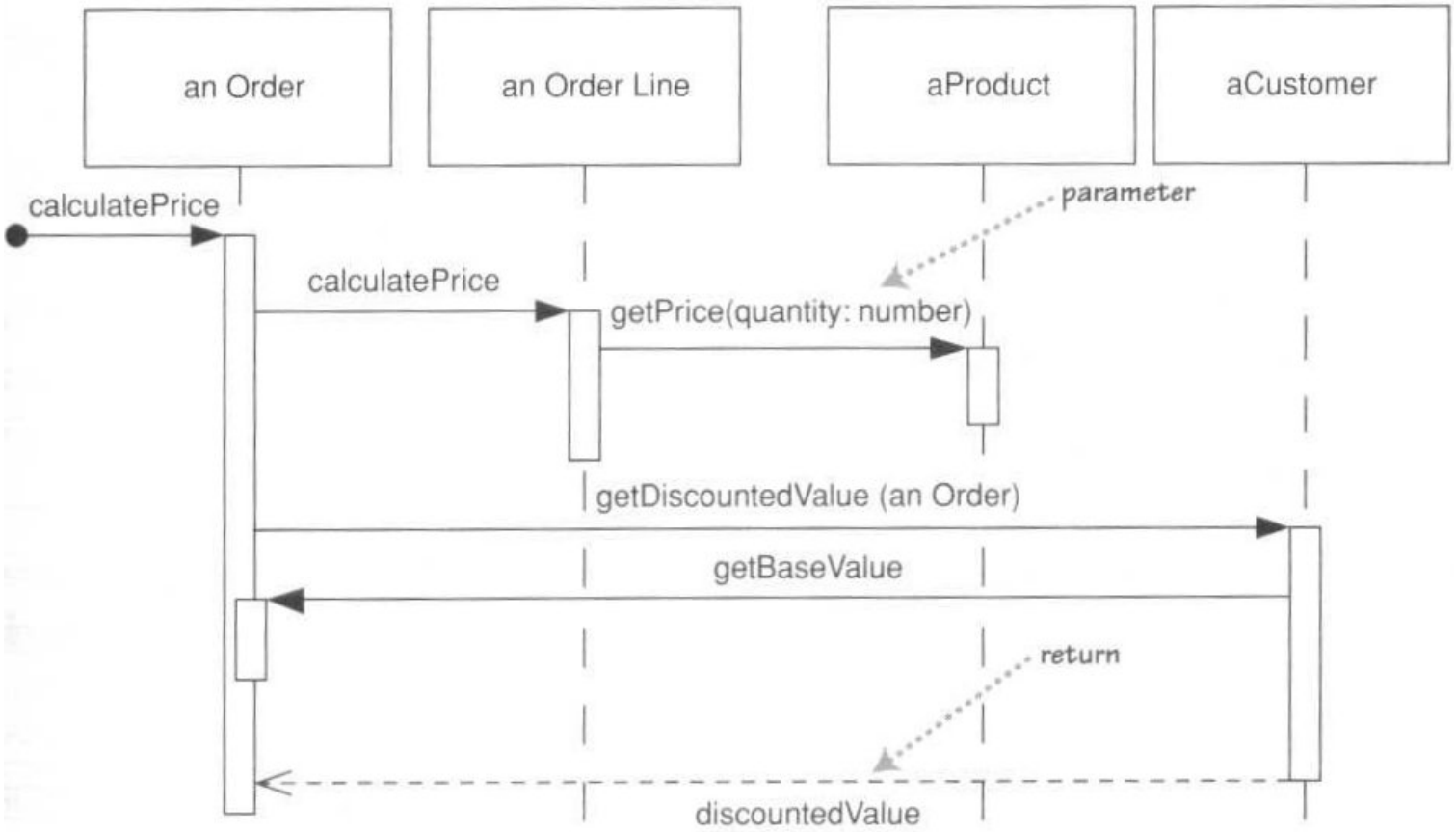
Centralised Control: Class diagram



- Only class Order has any interesting behaviour
- OrderLine, Customer, and Product are purely data classes (no methods)

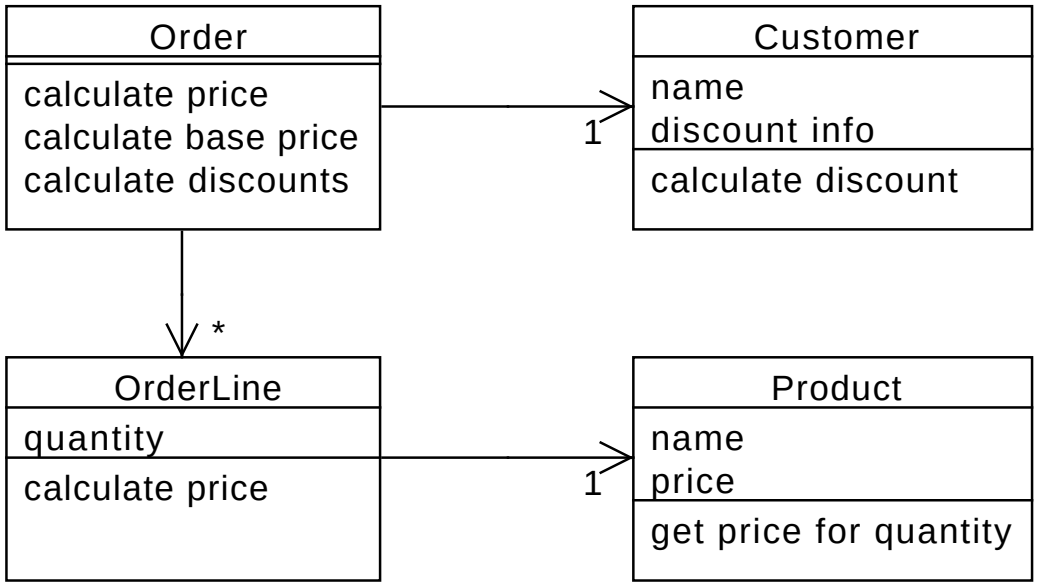
Computation is distributed among several objects (distributed control)

- The order computes the price by delegating part of the price calculation to order line and customer



→ distributed control

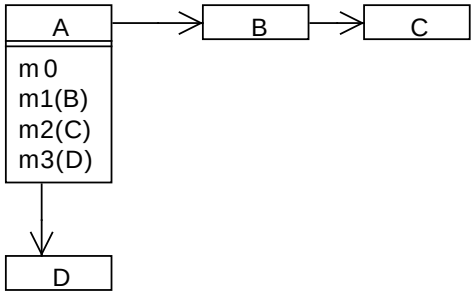
Distributed Control: Class diagram

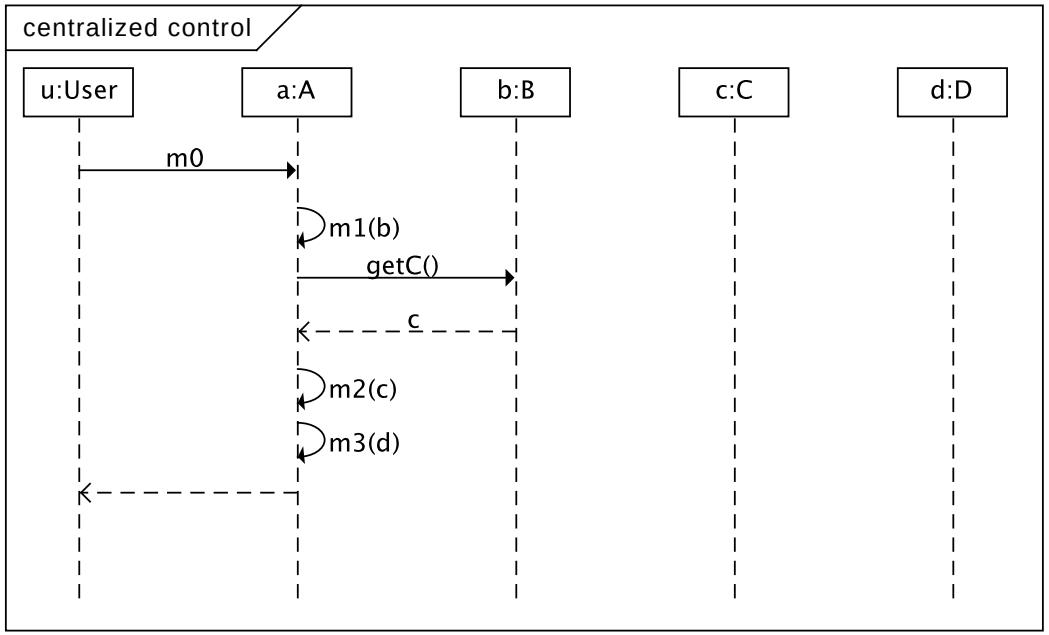


- More customer types can be added
- Each computing the discount differently

- The product now calculates the price depending on quantity
- One could now have products that are cheaper the more one buys

Design for change (I): Centralized Control

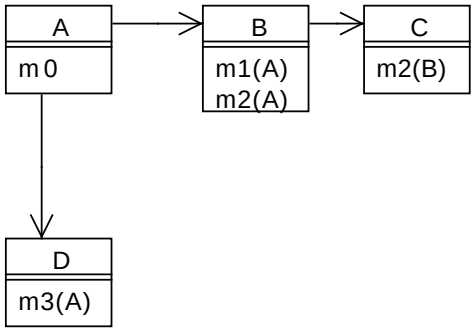


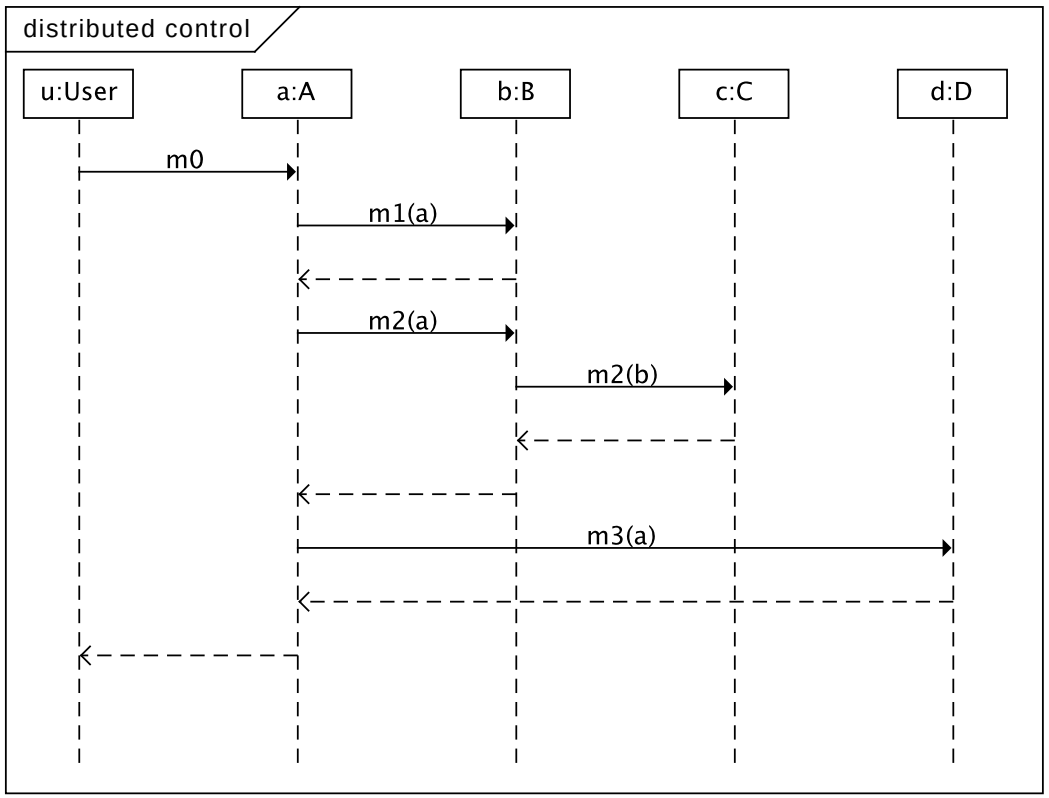



```
public class A {
   private B b;
   private D d;
   public void m0() {
      m1(b);
      C c = getC();
      m2(c);
      m3(d);
   }
   public void m1() {...}
   public void m2() {...}
   public void m3() {...}
}
```

Question: How easily can m_0 be adapted?

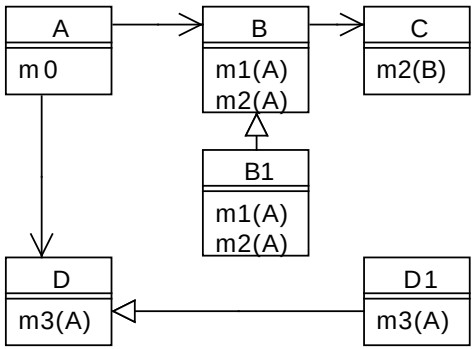
Design for change (II): Distributed Control

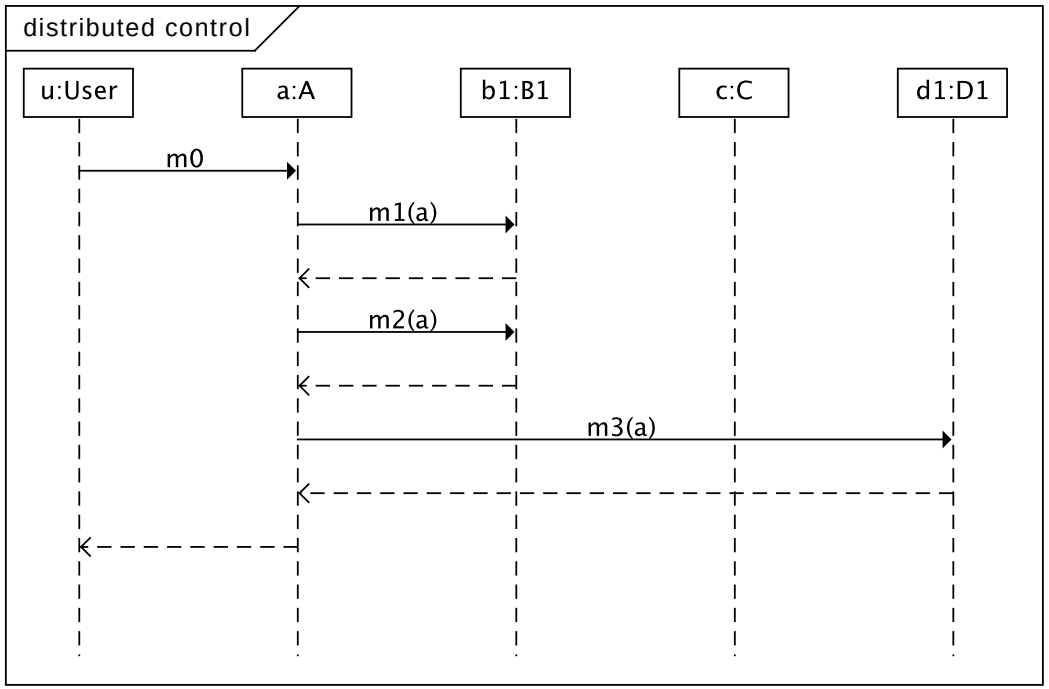





```
public class A {
    private B b;
    private D d;
    public void m0() {
        b.m1(this);
        b.m2(this);
        d.m3(this);
    }
}
public class B {
    private C c;
    public void m2(A a) {
        c.m2(this);
    }
}
```

Design for change (III): Distributed Control

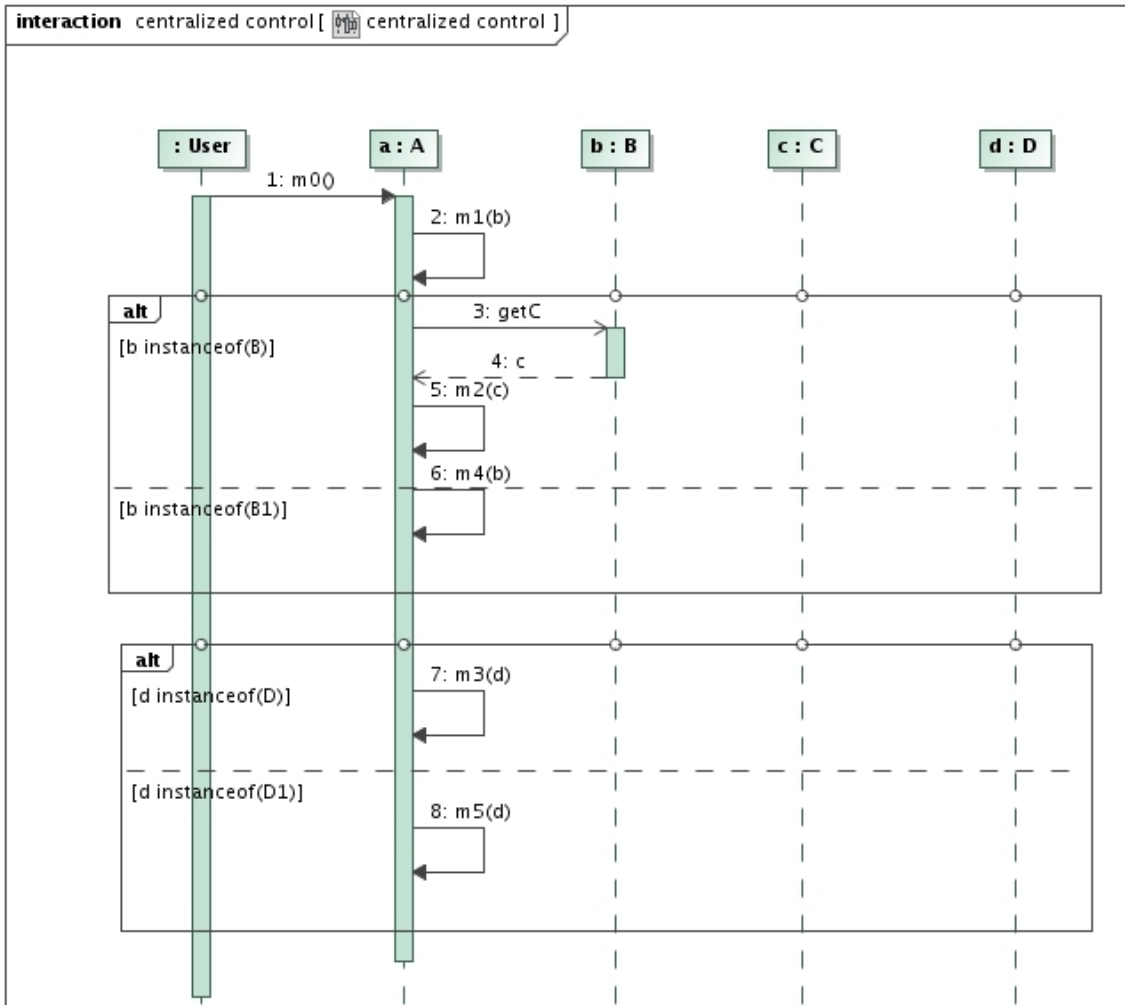




- The behaviour of m_0 can be adapted by using new subclasses of A, B, C, and D
- The subclasses have each their own version of m_0, m_1, m_2, and m_3
- What happens if a new subclass, e.g. D_2, is added?
- New subclasses can be easily added to adapt the behaviour, without m_0 having to change

→ The system can be used and adapted to situations which one has not thought of in the beginning

Design for change (IV): Centralized Control



- m_0 has to deal itself with all possible subclasses

→ use of `instanceof`

- What happens if a new subclass, e.g. D_2 is added to the system?

→ m_0 has to change

Centralised vs Distributed Control

- Centralised control
  - One method does all the work
  - The remaining objects are merely data objects and usually don't have their own behaviour
  - Typical for a procedural programming style
- Distributed control
  - Objects collaborate to achieve one task
  - Each object in a collaboration has behaviour (= is a "real" object)
  - Typical object-oriented style
    - Each object has its own responsibilities
    - Creates adaptable designs

Object-Orientation

Distributed Control is a characteristic of object orientation

Summary

- Activity Diagrams
- From requirements to design: CRC cards
- Sequence Diagrams
- Object Orientation: Centralized vs Decentralized control

Software Engineering I (02161)

Week 5

Assoc. Prof. Hubert Baumeister

Informatics and Mathematical Modelling
Technical University of Denmark

Spring 2011