Software Engineering I (02161)

Week 5

Assoc. Prof. Hubert Baumeister

DTU Compute
Technical University of Denmark

Spring 2015

Contents

From Requirements to Design I

Design process (abstract)

1. Choose a set of user stories to implement
2. Select the user story with the highest priority
   a. Design the system by executing the user story in your head
      → e.g. use CRC cards for this
   b. Extend an existing class diagram with classes, attributes, and methods + Sequence diagram
   c. Create acceptance tests
   d. Implement the user story test-driven, creating tests as necessary and guided by your design
3. Repeat step 2 with the user story with the next highest priority

From Requirements to Design II

Model first

1. Choose a set of user stories to model
   - → Select those that **define** the architecture of the system
2. Select the user story with the highest priority
   a. Design the system by executing the user story in your head
      - → e.g. use CRC cards for this
   b. **Extend** the existing class diagram with classes, attributes, and methods + Sequence diagrams
3. Repeat step 2 with the user story with the next highest priority

From Requirements to Design II (cont.)

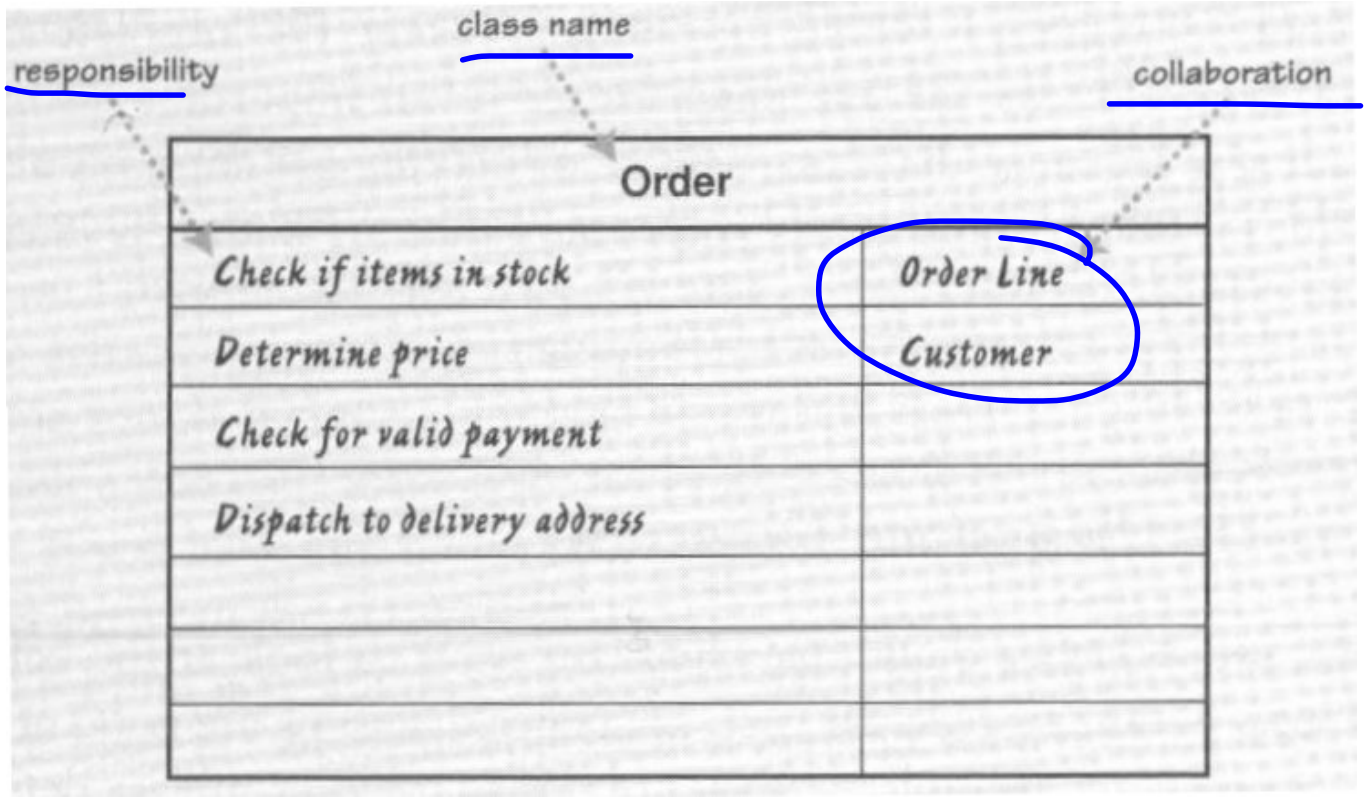
Implement the model

1. Choose a set of user stories to implement
2. Select the user story with the highest priority
   - c. Create acceptance tests
   - d. Implement the user story test-driven, creating additional tests as necessary and guided by your design
     - → based on the classes, attributes, and methods of the model
     - → implement **only** the classes, attributes, and methods needed to implement the user story
     - → Critera: 100% coverage of the code based on the tests you have
3. Repeat step 2 with the user story with the next highest priority

Introduction CRC Cards

- Class Responsibility Collaboration
- Developed in the 80's
- Used to
  - Analyse a problem domain
  - Discover object-oriented design
  - Teach object-oriented design
- Object-oriented design:
  - Objects have state and behaviour
  - Objects delegate responsibilities
  - "Think objects"

CRC Card Template



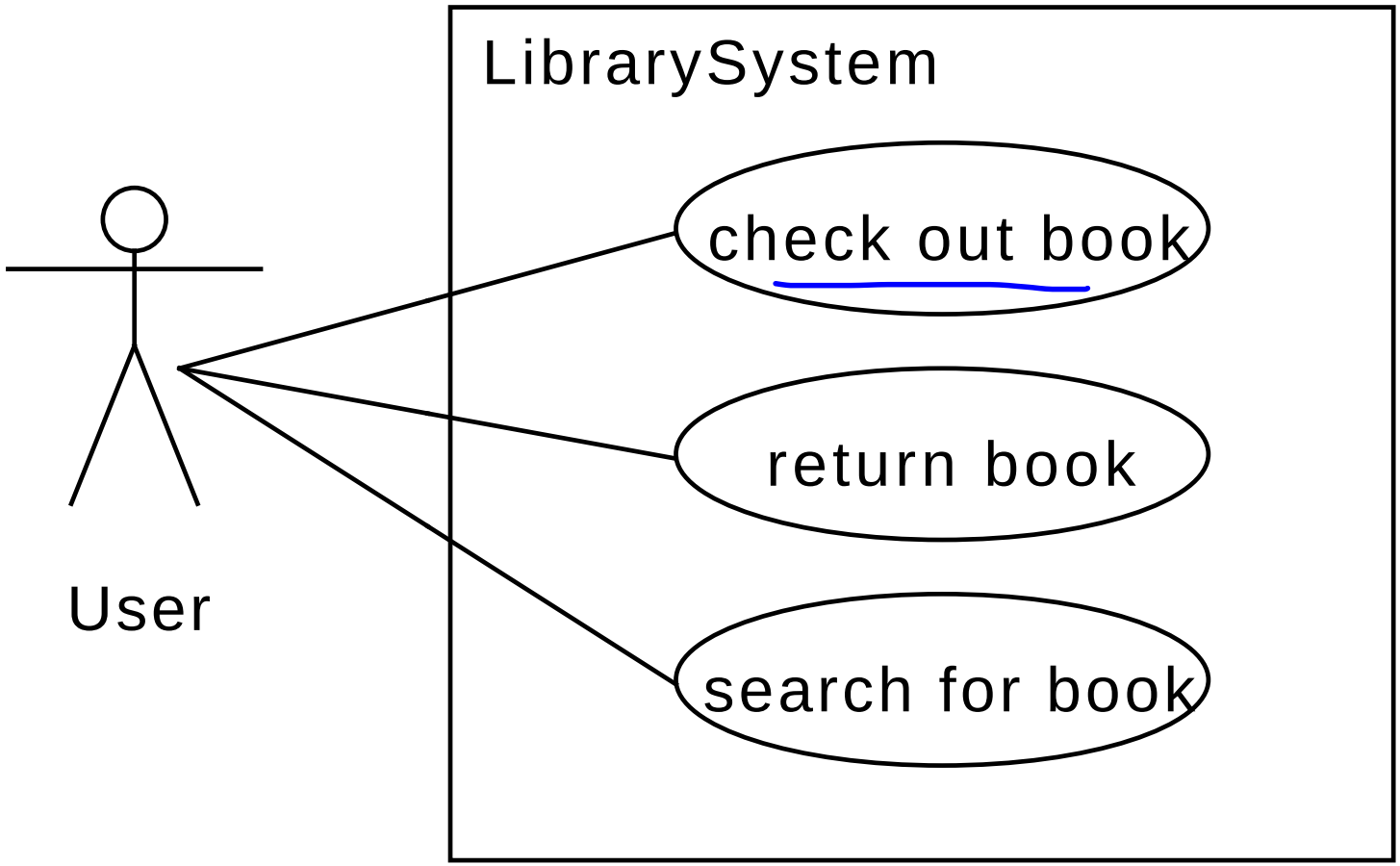
A larger example

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

Process

- Basic: Simulate the execution of use case scenarios / user stories
- Steps
  1. Brainstorm classes/objects/components
  2. Assign classes/objects/components to persons (group up to 6 people)
  4. Execute the scenarios one by one
     a) add new classes/objects/components as needed
     b) add new responsibilities
     c) delegate to other classes / persons

Library Example: Use Case Diagram



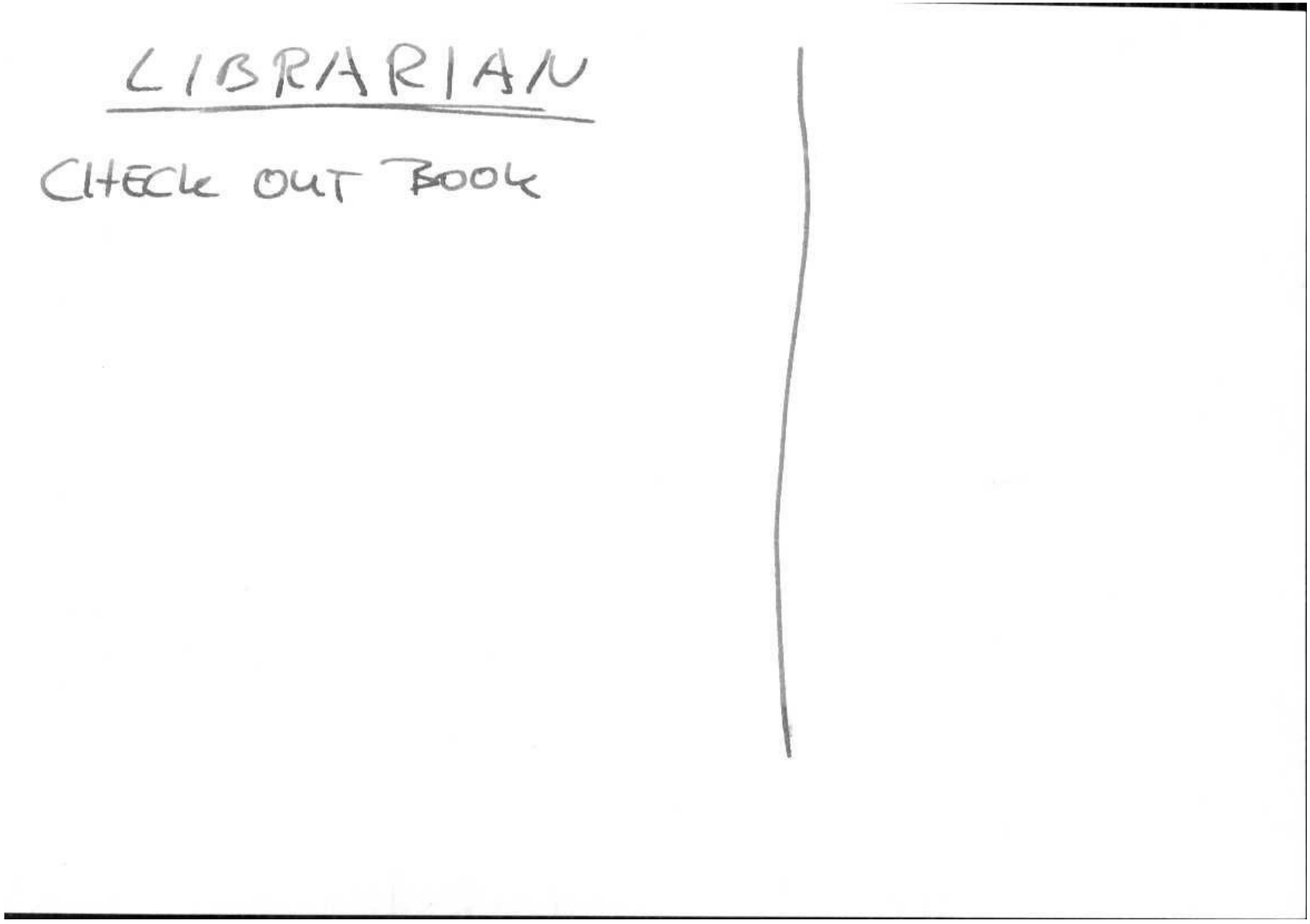
Library Example: Detailed Use Case *Check Out Book*

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

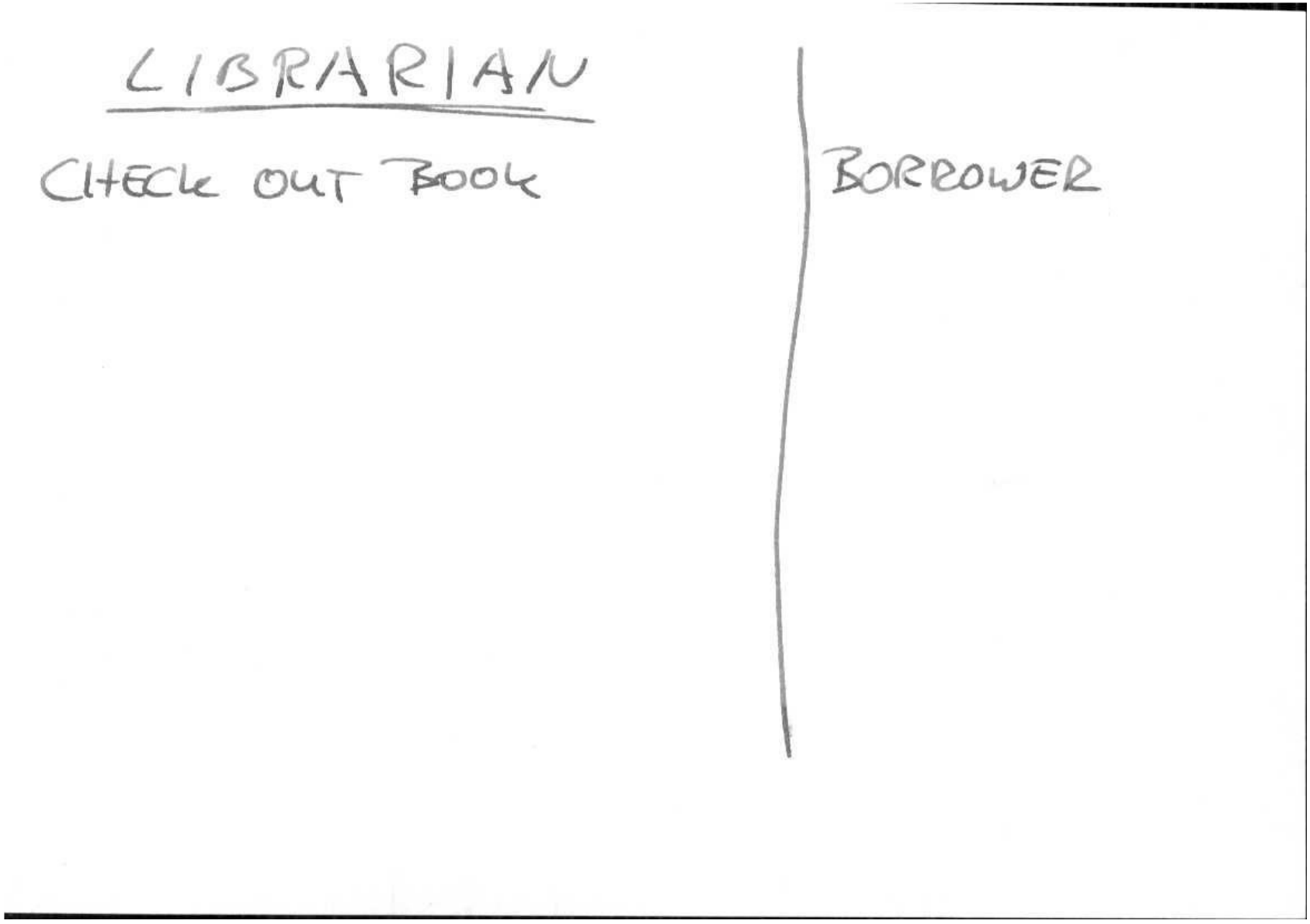
Example II

- Set of initial CRC cards: Librarien, Borrower, Book
- Use case **Check out book** main scenario (user story)
  - ”What happens when Barbara Stewart, who has no accrued fines and one outstanding book, not overdue, checks out a book entitled Effective C++ Strategies+?”

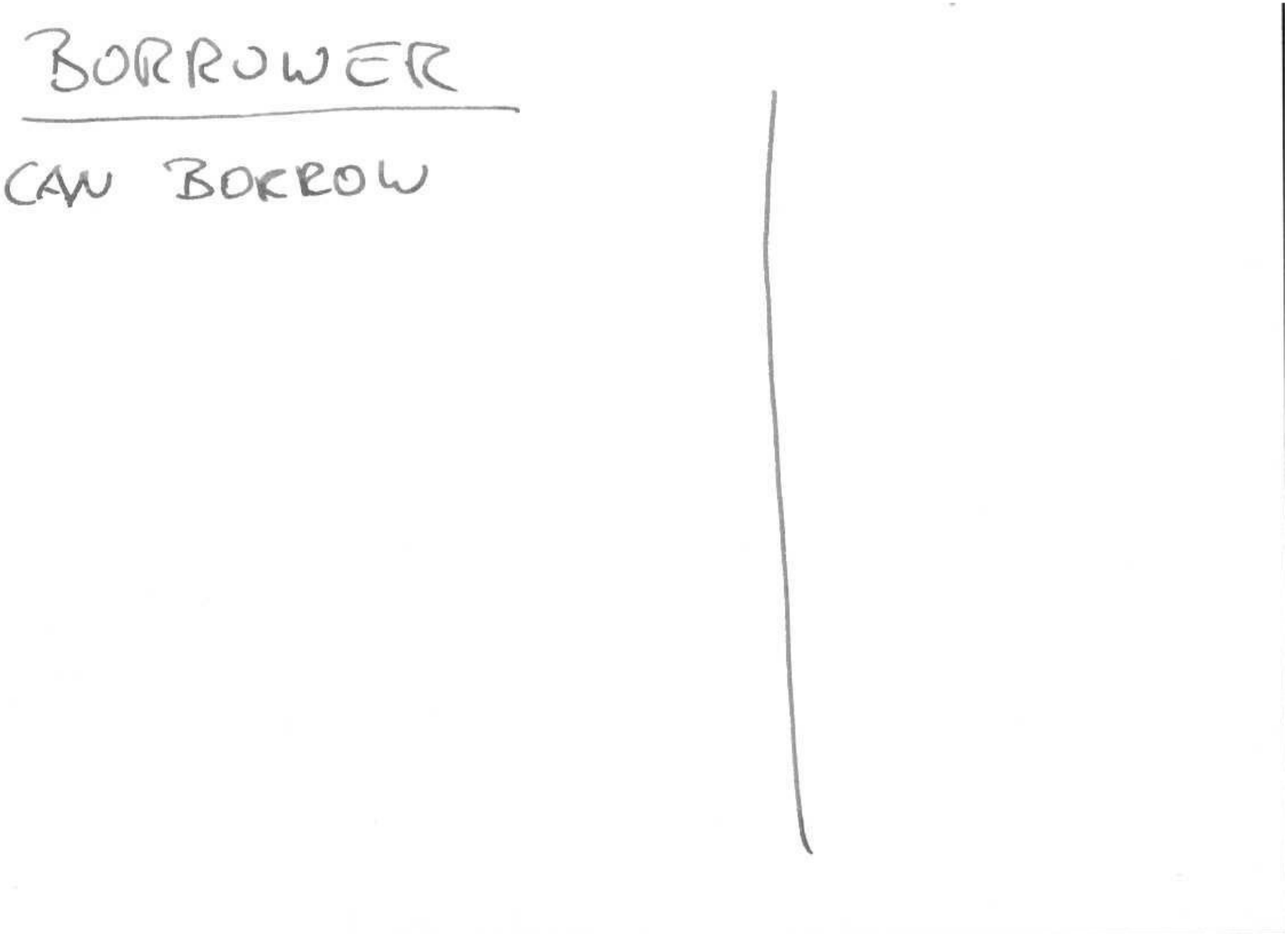
Library Example: CRC cards



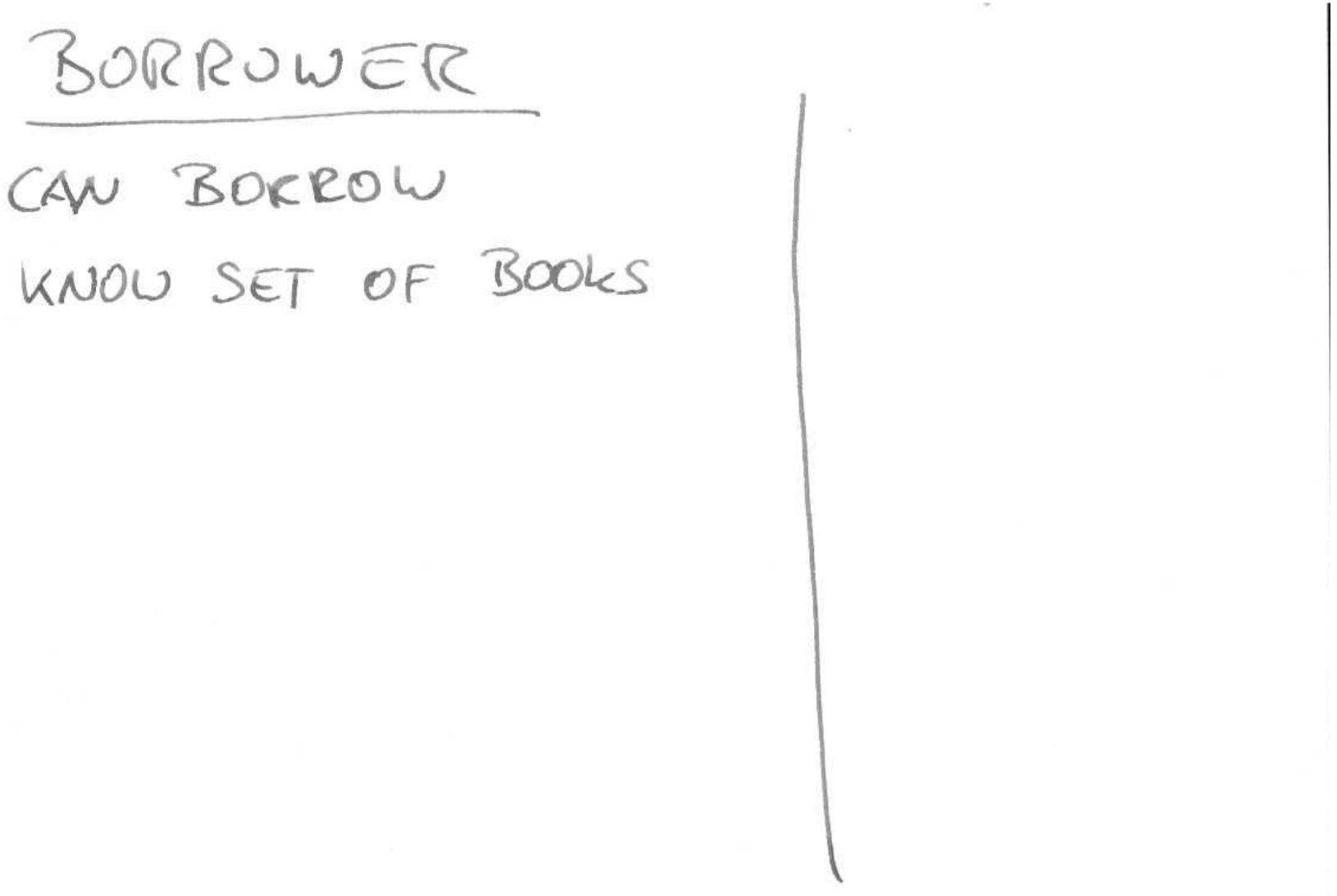
Library Example: CRC cards



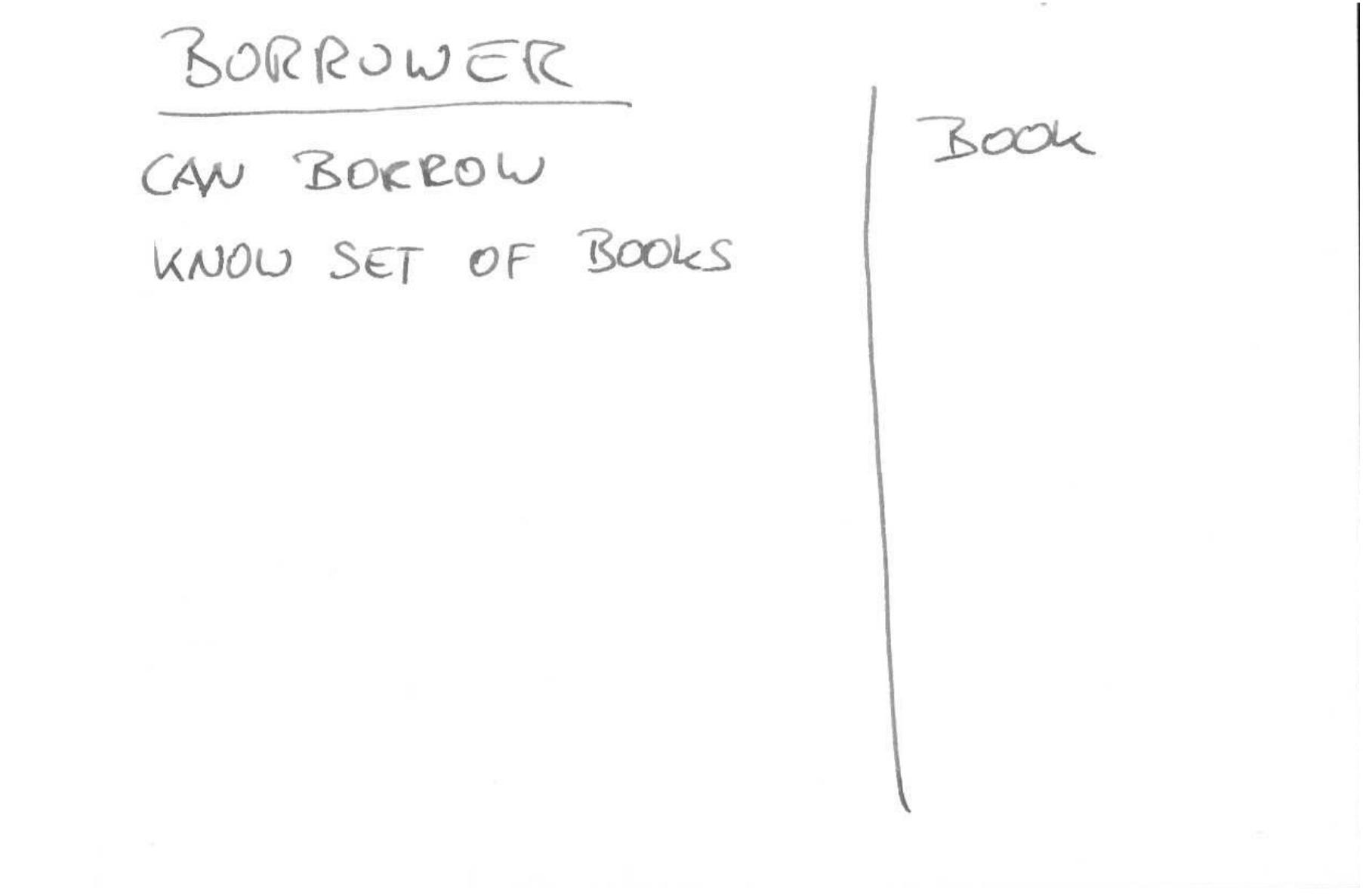
Library Example: CRC cards



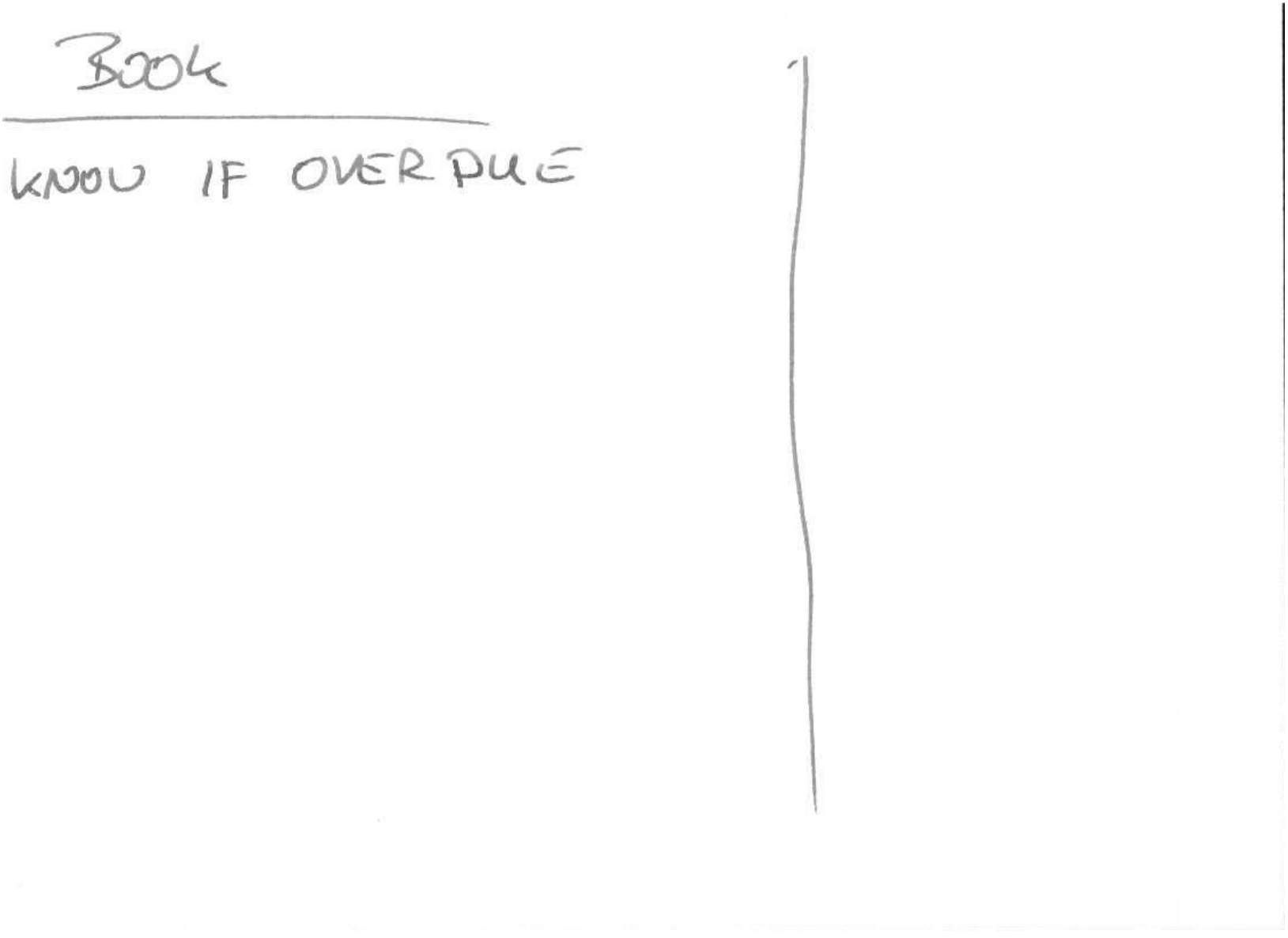
Library Example: CRC cards



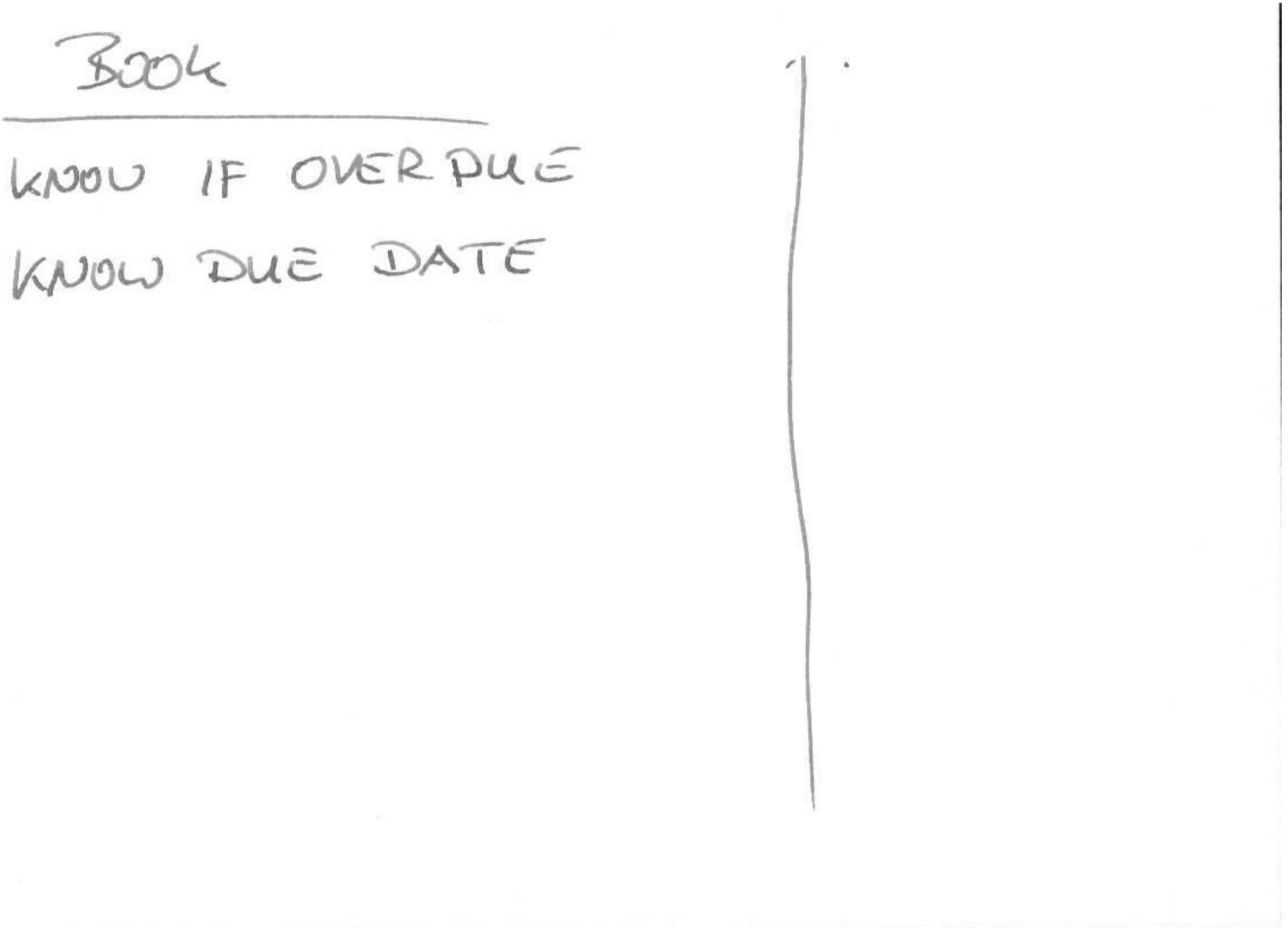
Library Example: CRC cards



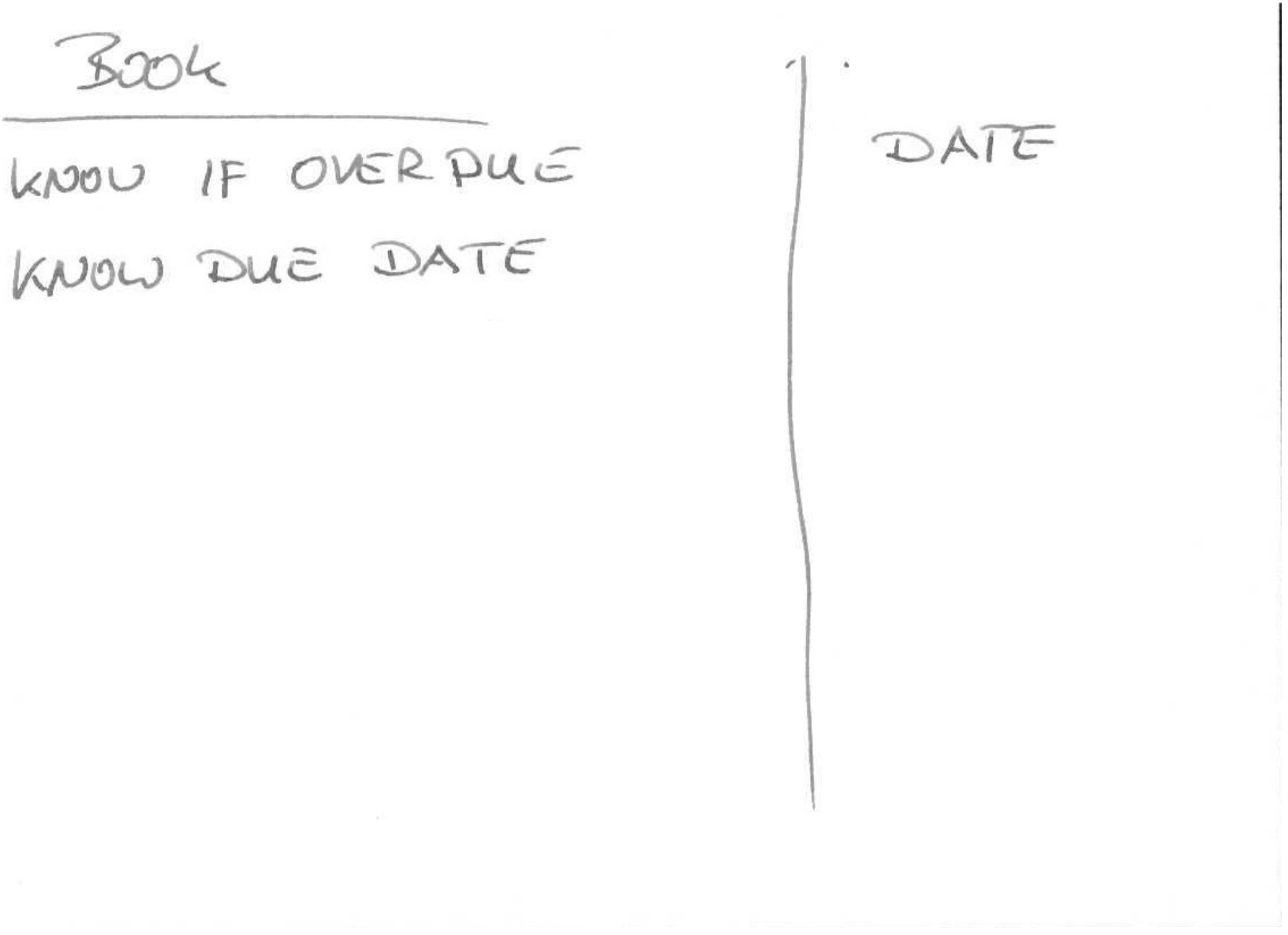
Library Example: CRC cards



Library Example: CRC cards

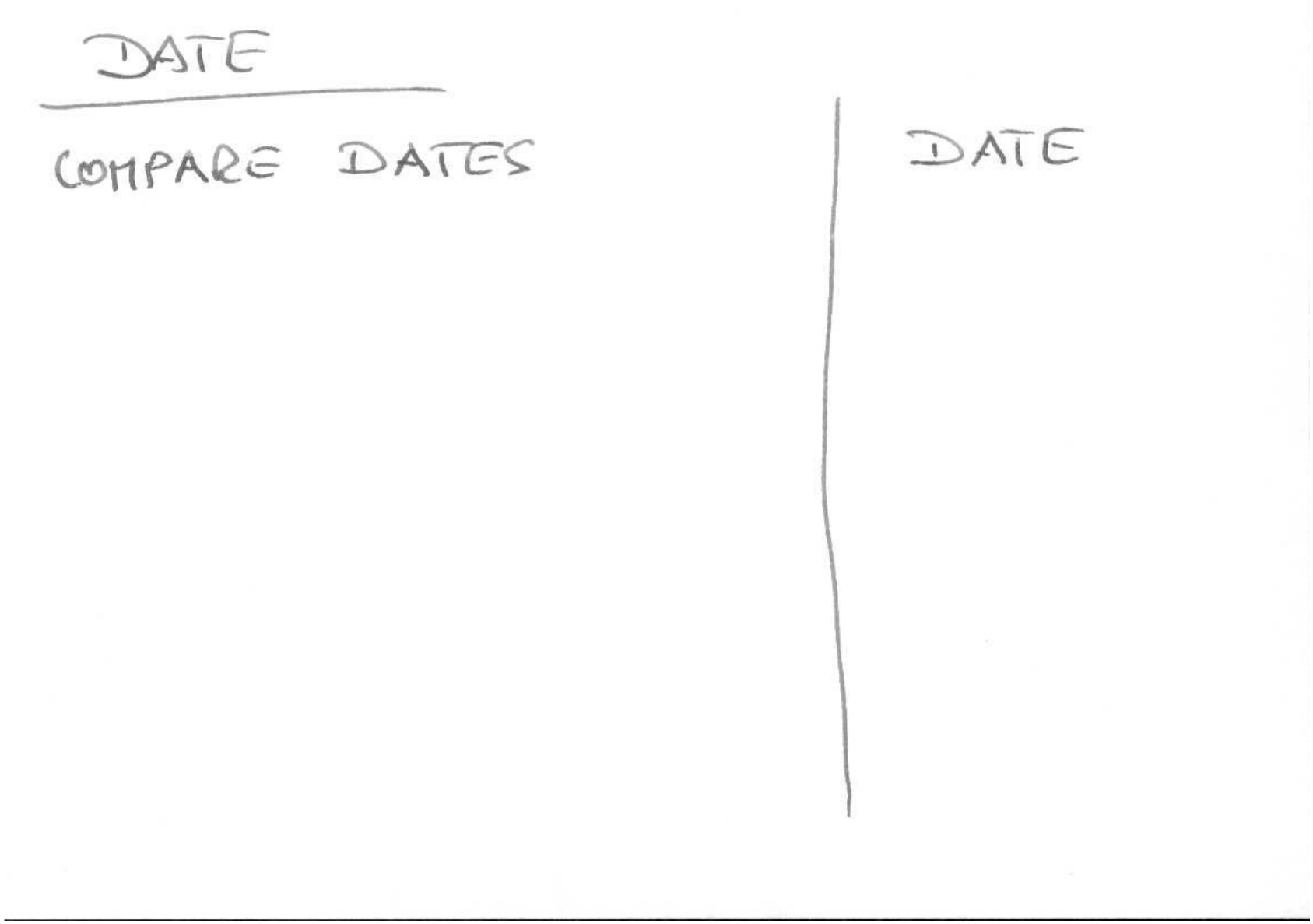


Library Example: CRC cards

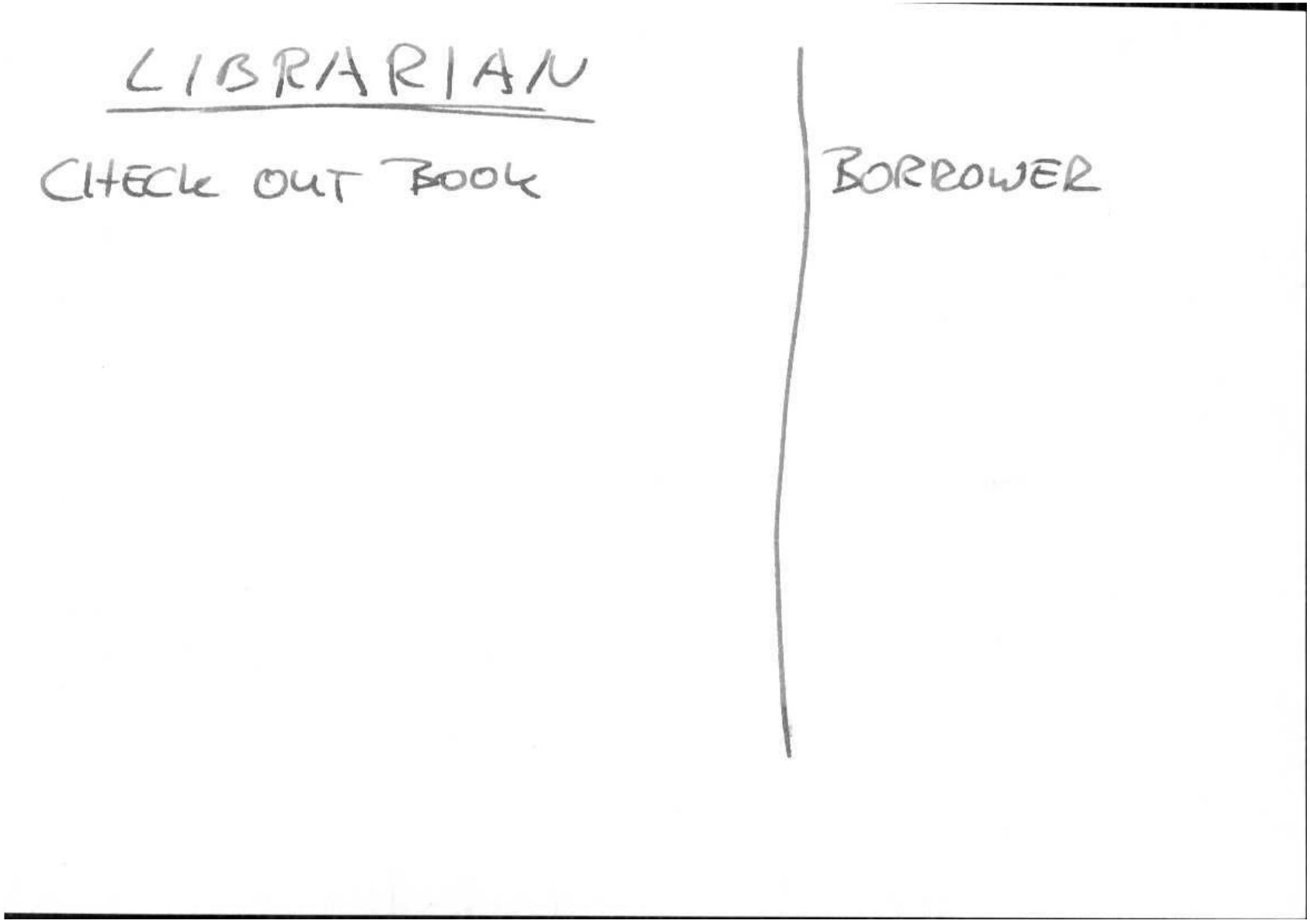


Library Example: CRC cards

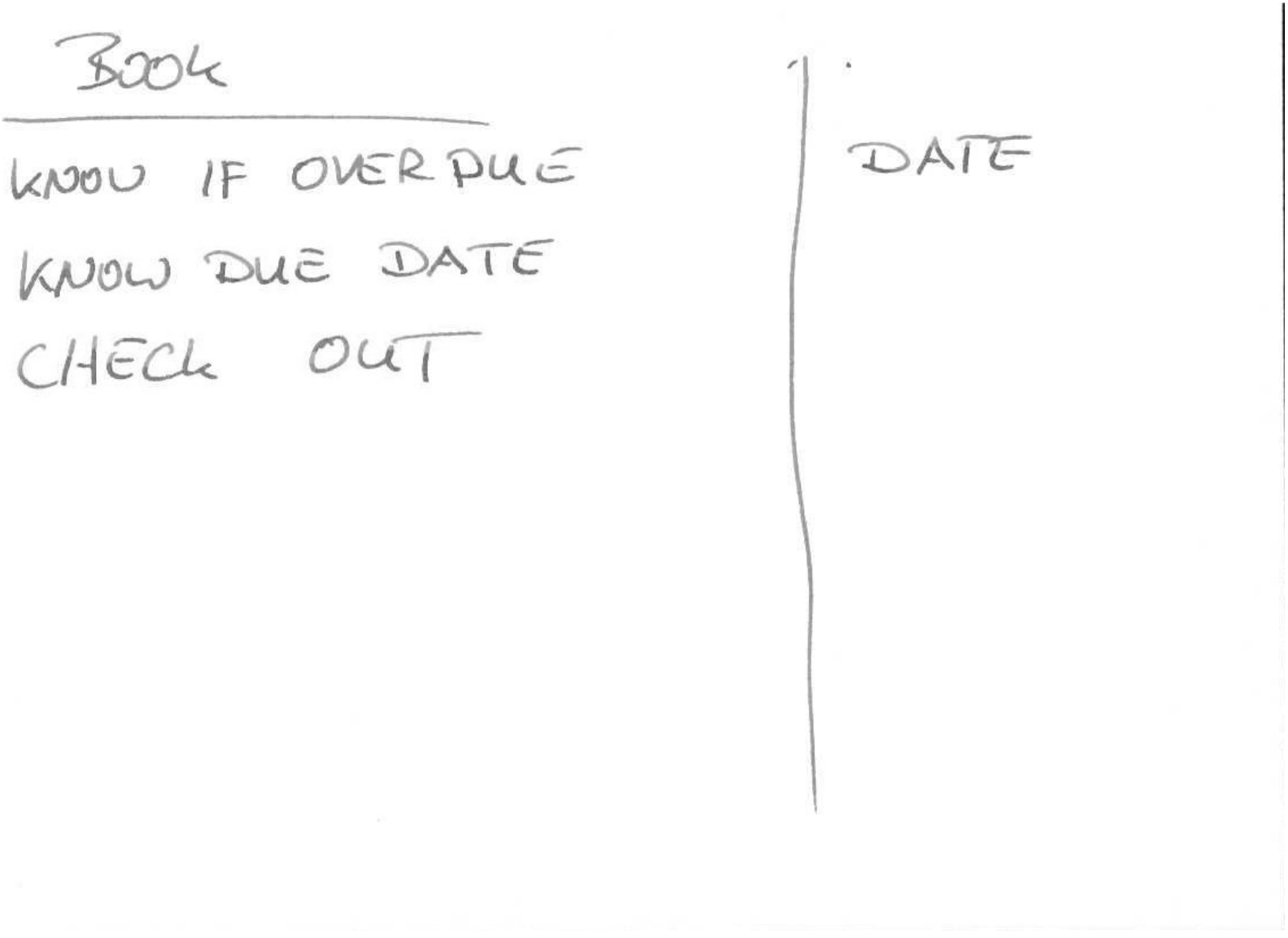
Library Example: CRC cards



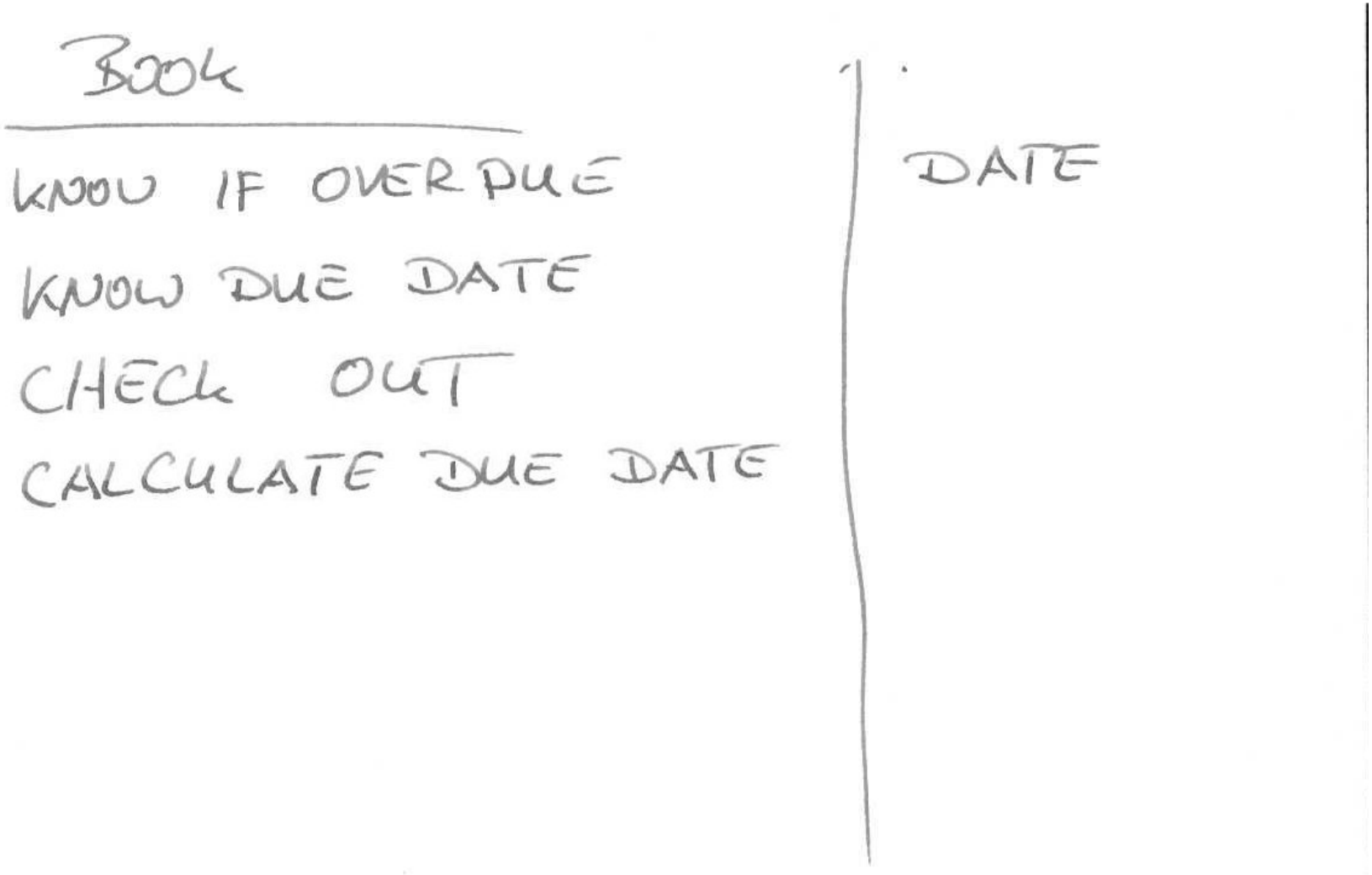
Library Example: CRC cards



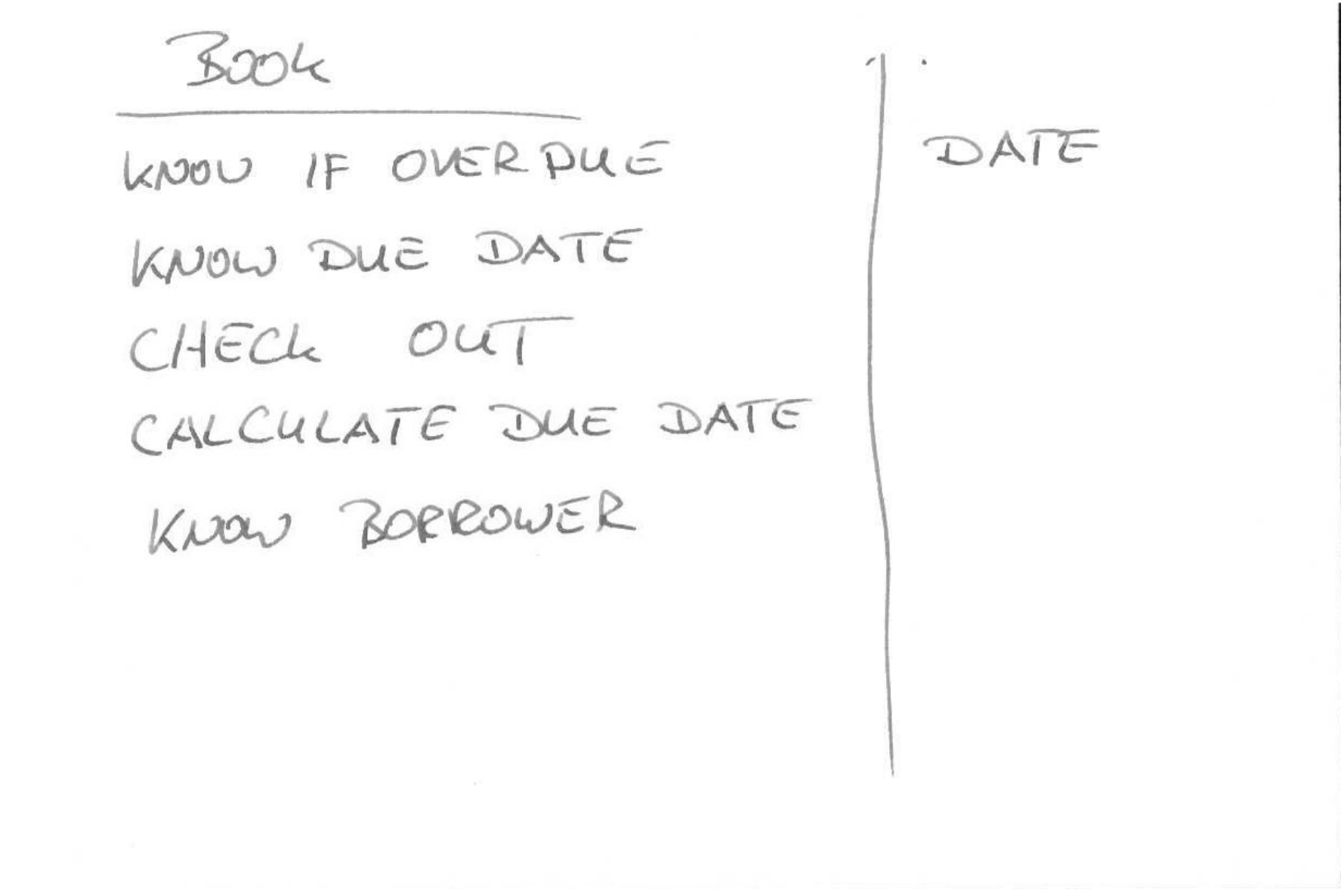
Library Example: CRC cards



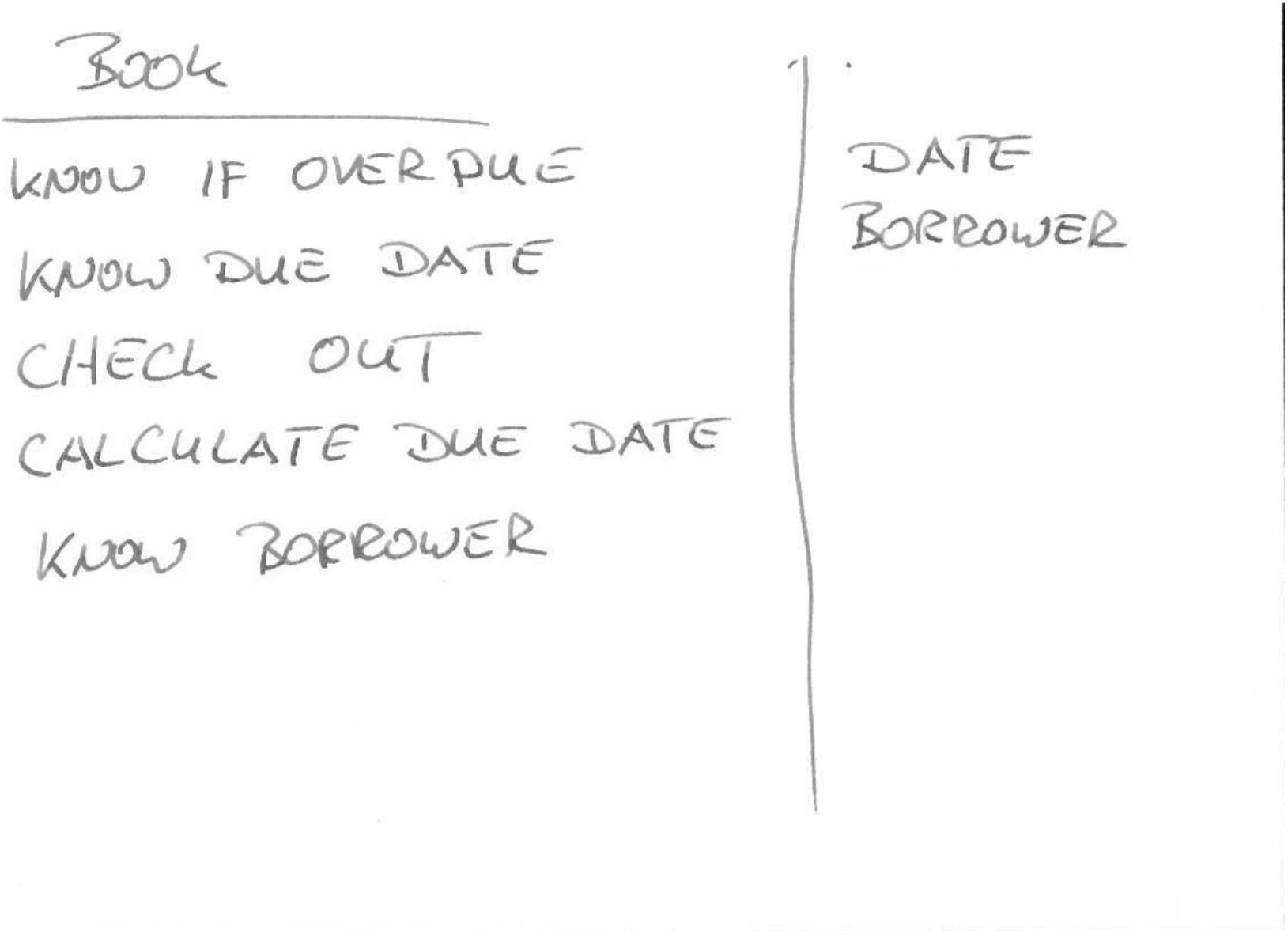
Library Example: CRC cards



Library Example: CRC cards



Library Example: CRC cards

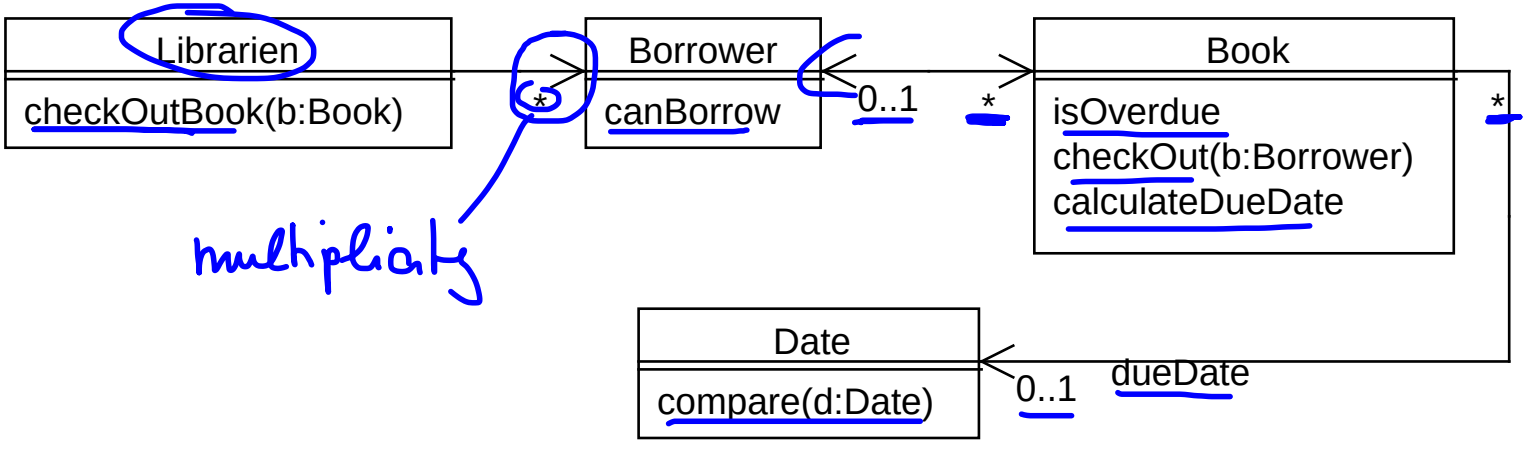


Library Example: All CRC cards

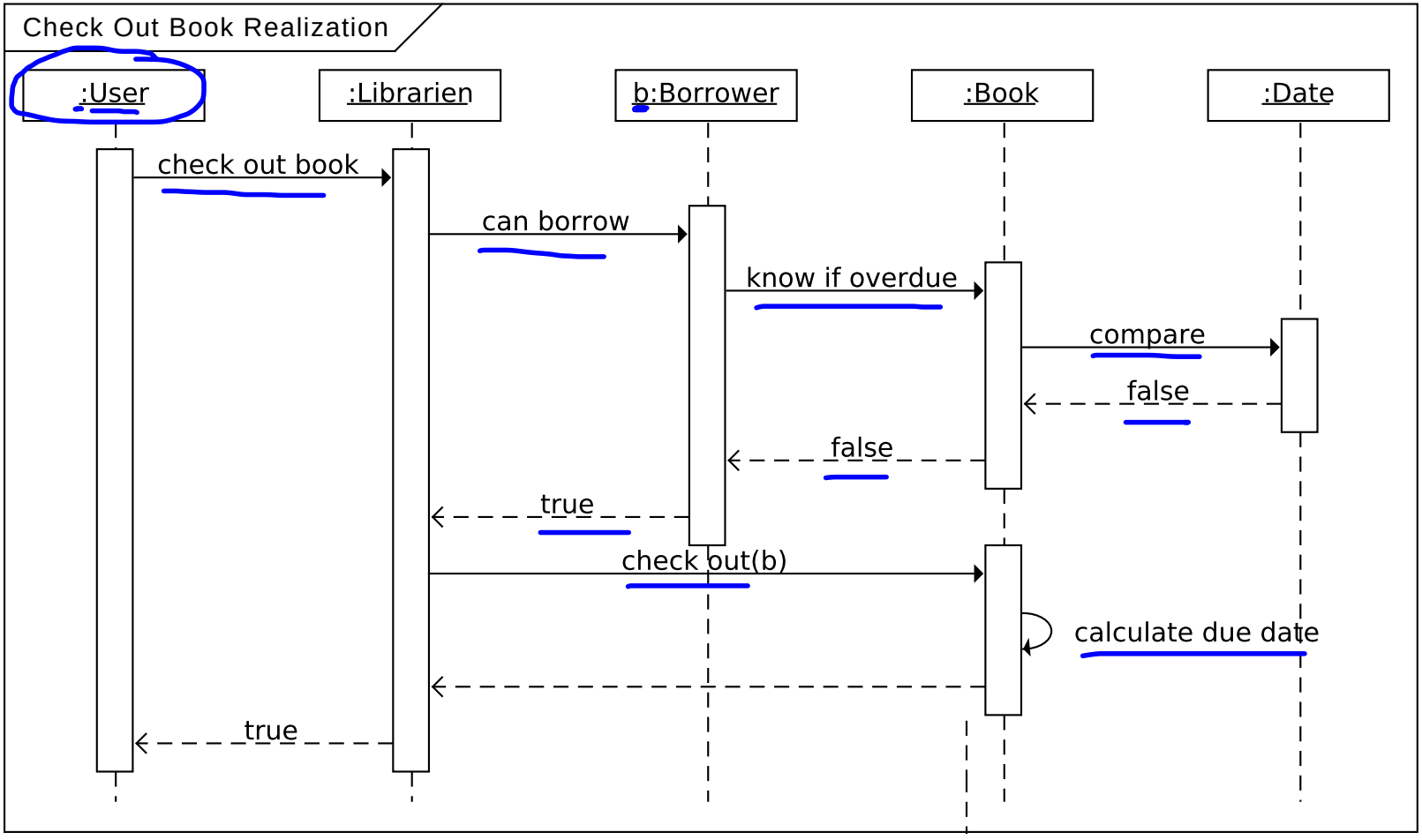
Process: Next Steps

- Review the result
  - Group cards
  - Check cards
  - Refactor
- Transfer the result
  - Implement the design test-driven
  - UML model

Example: Class Diagram (so far)



Example: Sequence Diagram for Check-out book



Alternative

- Build class and sequence diagrams directly
  - Danger: talk about the system instead of being part of the system
  - Possible when object-oriented principles have been learned
  - CRC cards help with object-oriented thinking

Contents

UML

- Unified Modelling Language (UML)
- Set of graphical notations: class diagrams, state machines, sequence diagrams, activity diagrams, ...
- Developed in the 90's
- ISO standard

Class Diagram

- Structure diagram of object oriented systems
- Possible level of details

Domain Modelling : typically low level of detail

⋮

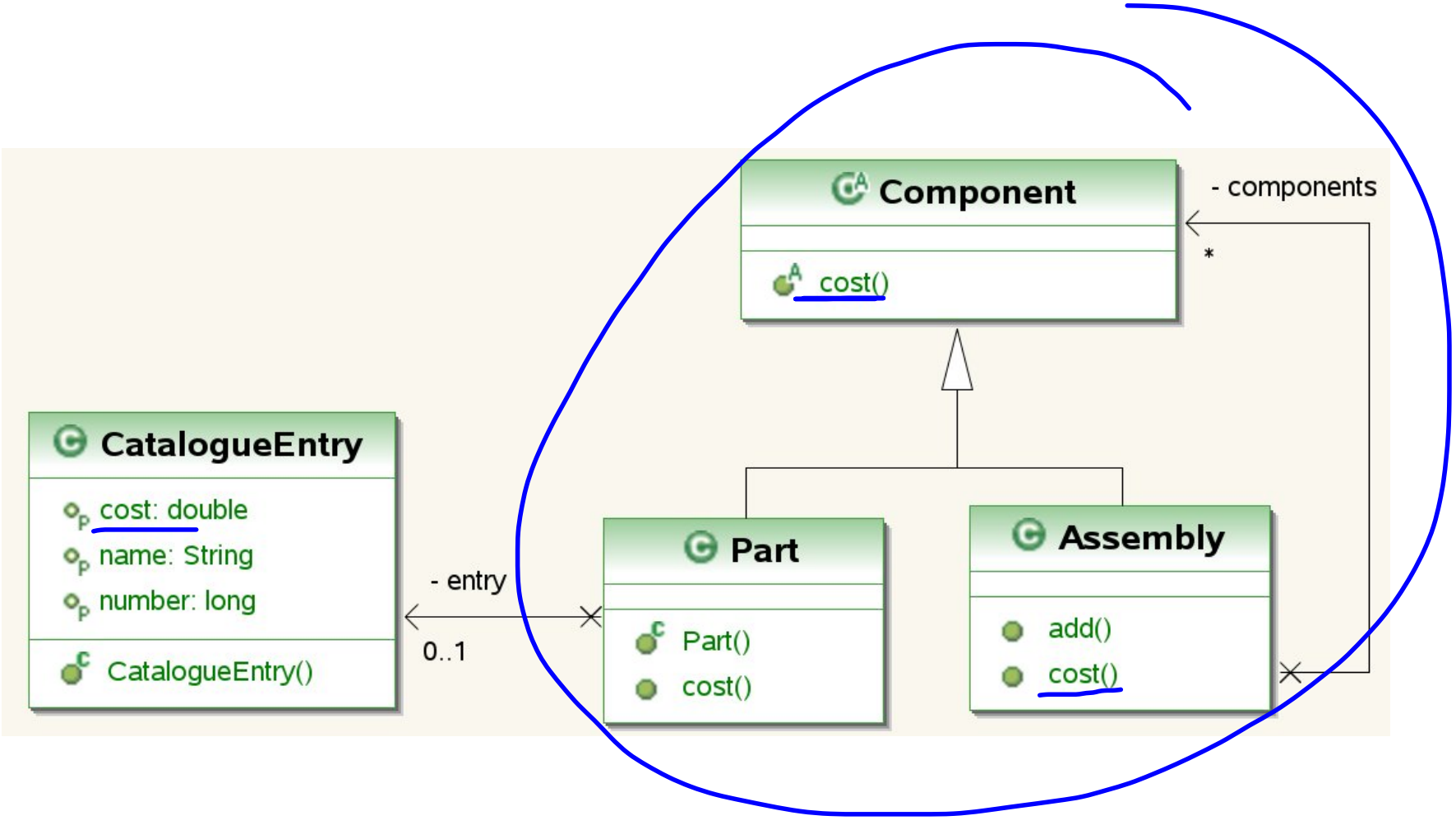
Implementation : typically high level of detail

Why a graphical notation?

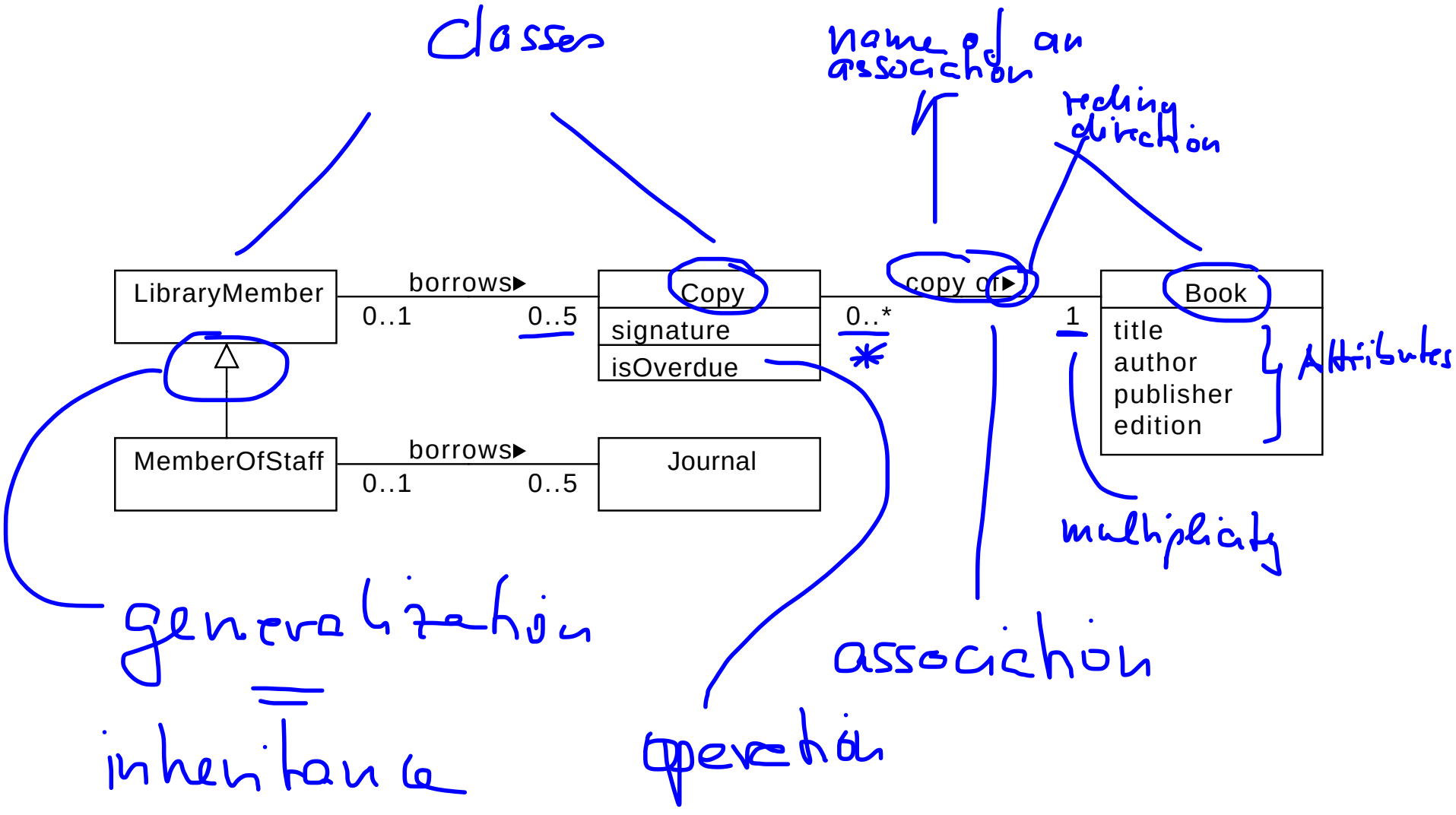
```
public class Assembly
            extends Component {
 public double cost() {   }
 public void add(Component c) {}
 private Collection<Component>
      components;
}

public class CatalogueEntry {
 private String name = "";
 public String getName() {}
 private long number;
 public long getNumber() {}
 private double cost;
 public double getCost() {}
}
```

```
public abstract class Component {
 public abstract double cost();
}

public class Part extends Component
 private CatalogueEntry entry;
 public CatalogueEntry getEntry() {}
 public double cost(){}
 public Part(CatalogueEntry entry){}
```

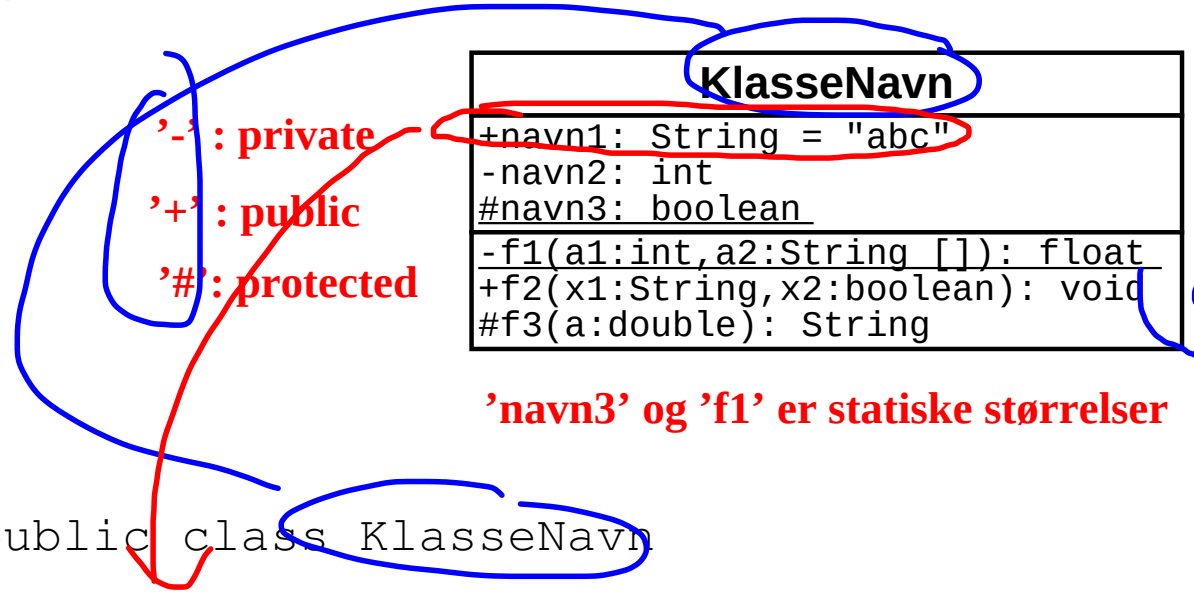
Why a graphical notation?



Class Diagram Example



General correspondence between Classes and Programs



underline static method or static attribute

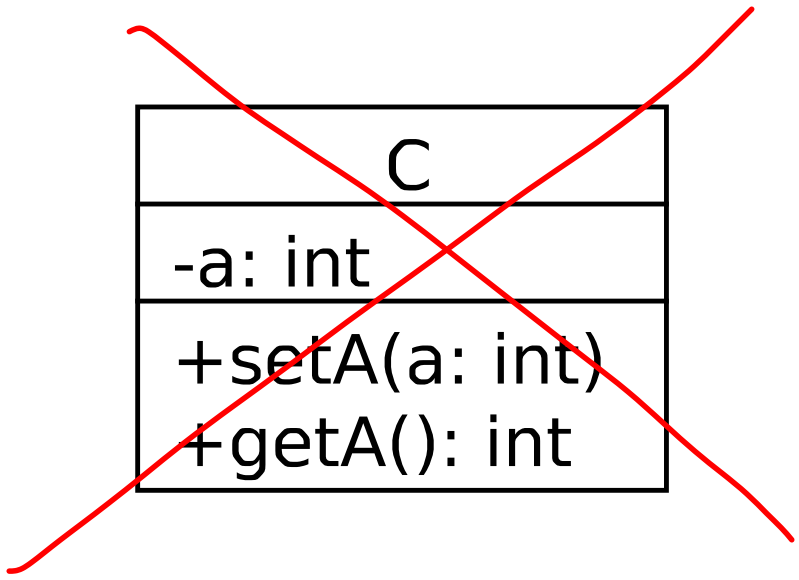
```
public class KlasseNavn
{
   private String navn1 = "abc";
    private int navn2;
    protected static boolean navn3;

    private static float f1(int a1, String[] a2) { ... }
    public void f2(String x1, boolean x2) { ... }
    protected String f3(double a) { ... }
    public String getNavn1(); {...}
    public void setNavn1(String n) {...}
}
```

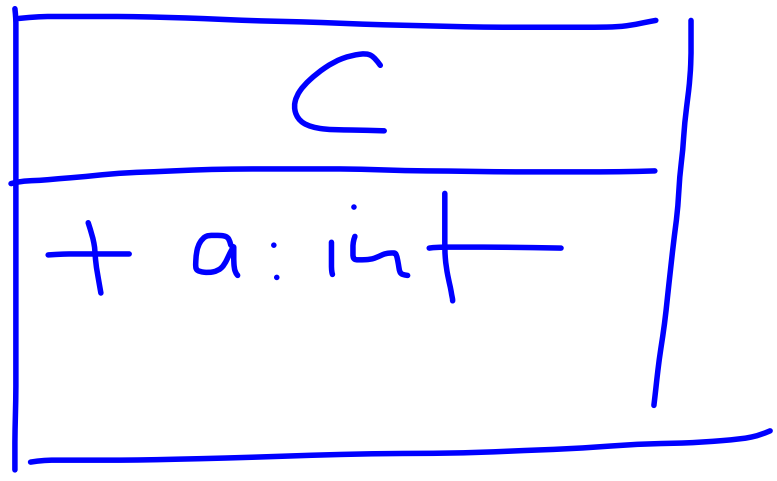
Class Diagram and Program Code

```
public class C {
  private int a;
  public int getA() { return a; }
  public void setA(int a) { this.a = a; }
}
```

Class Diagram and Program Code

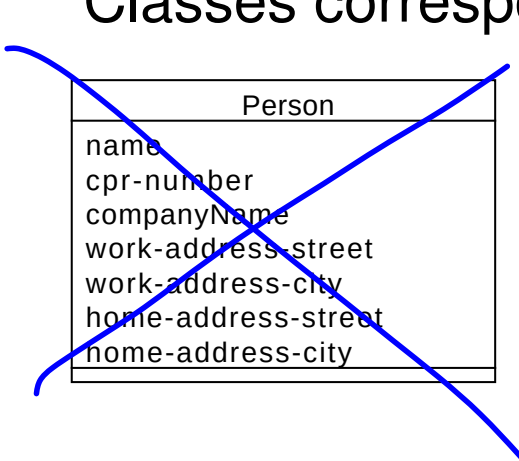
```
public class C {
  private int a;
  public int getA() { return a; }
  public void setA(int a) { this.a = a; }
}
```

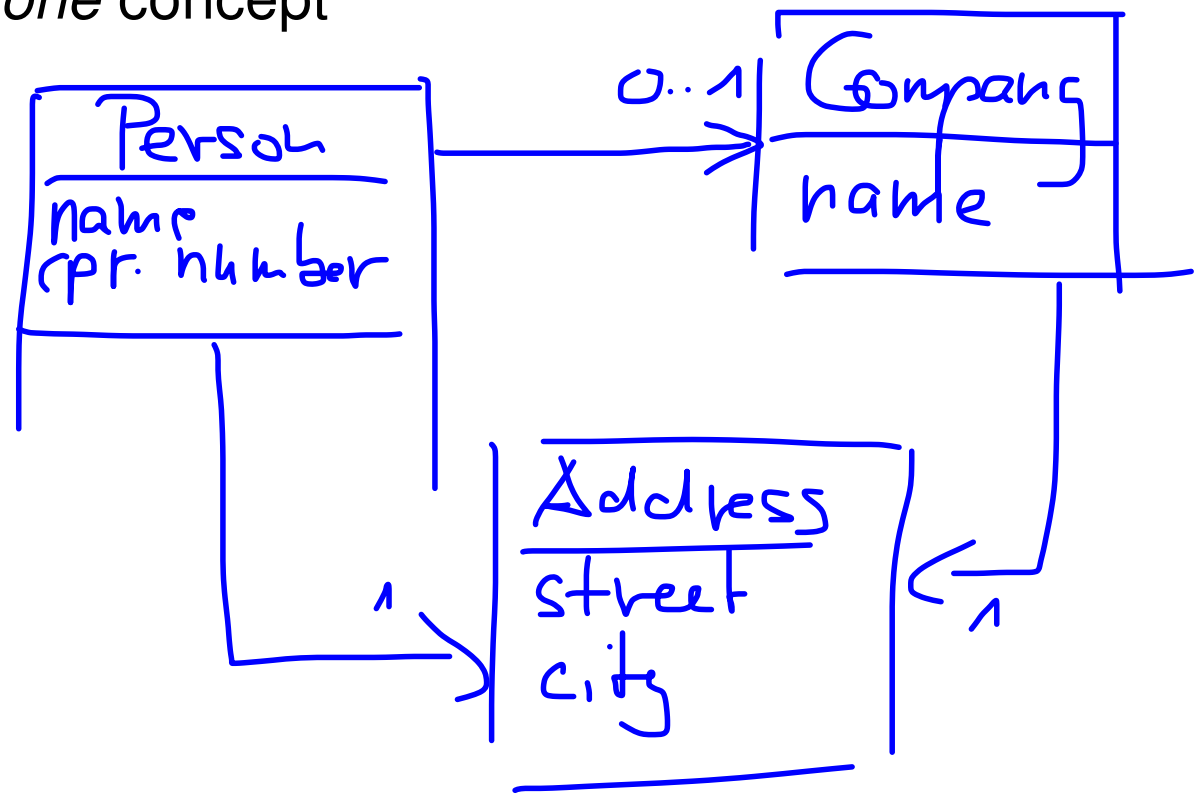




Classes

Classes corresponds to *one* concept



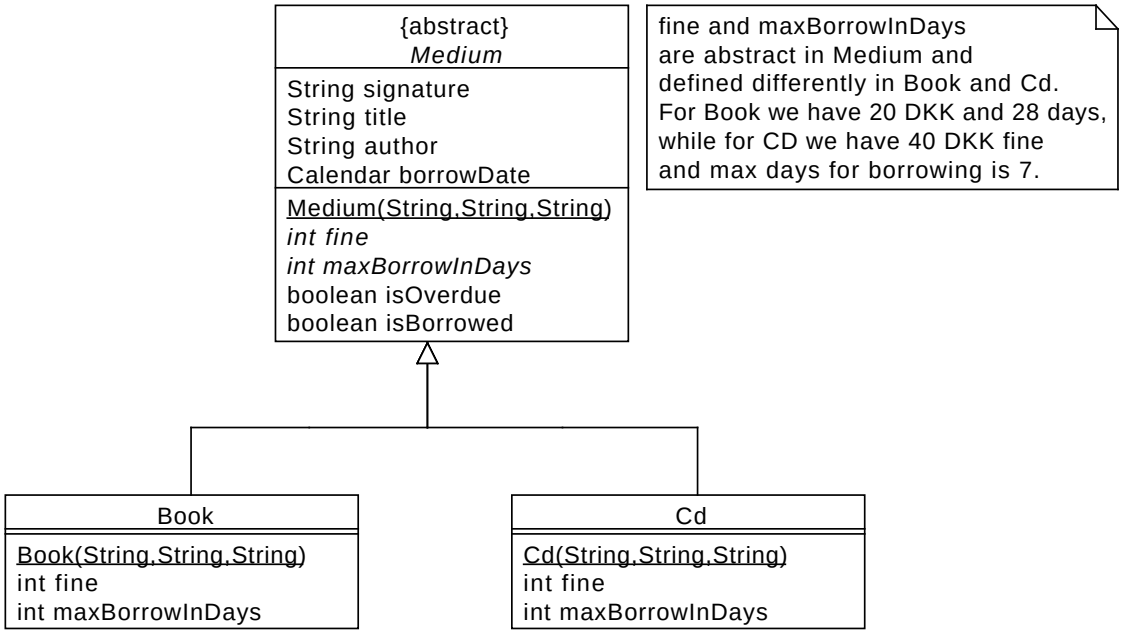


Generalization / Inheritance

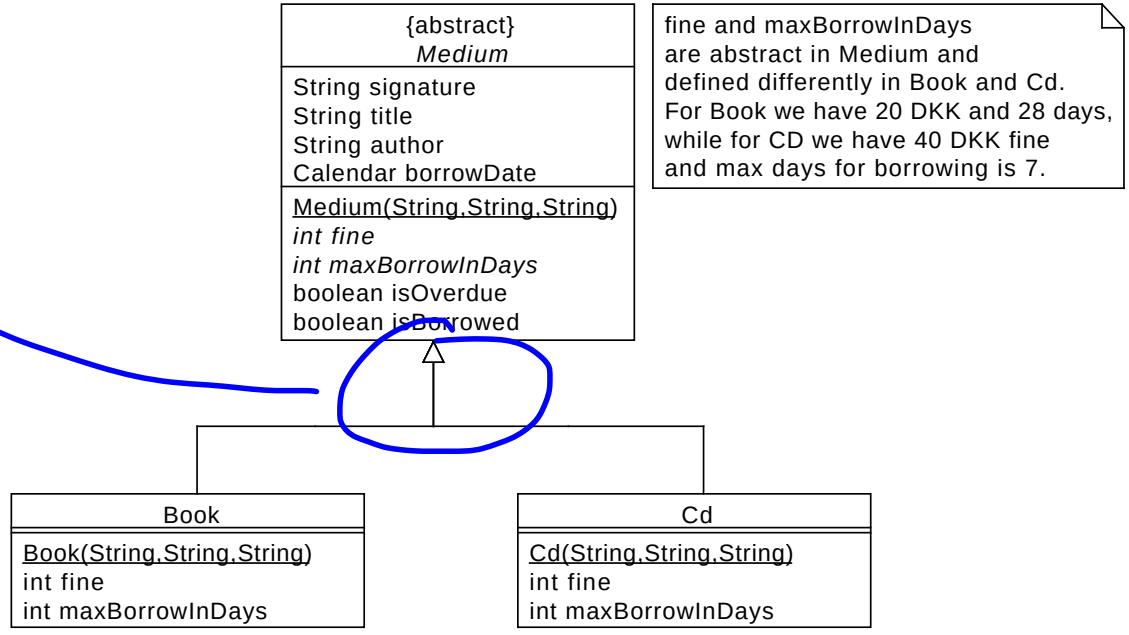
- Programming languages like Java: Inheritance

```
abstract public class Medium { ... }
public class Book extends Medium { ... }
public class Cd extends Medium { ... }
```

- UML: Generalization / Specialization



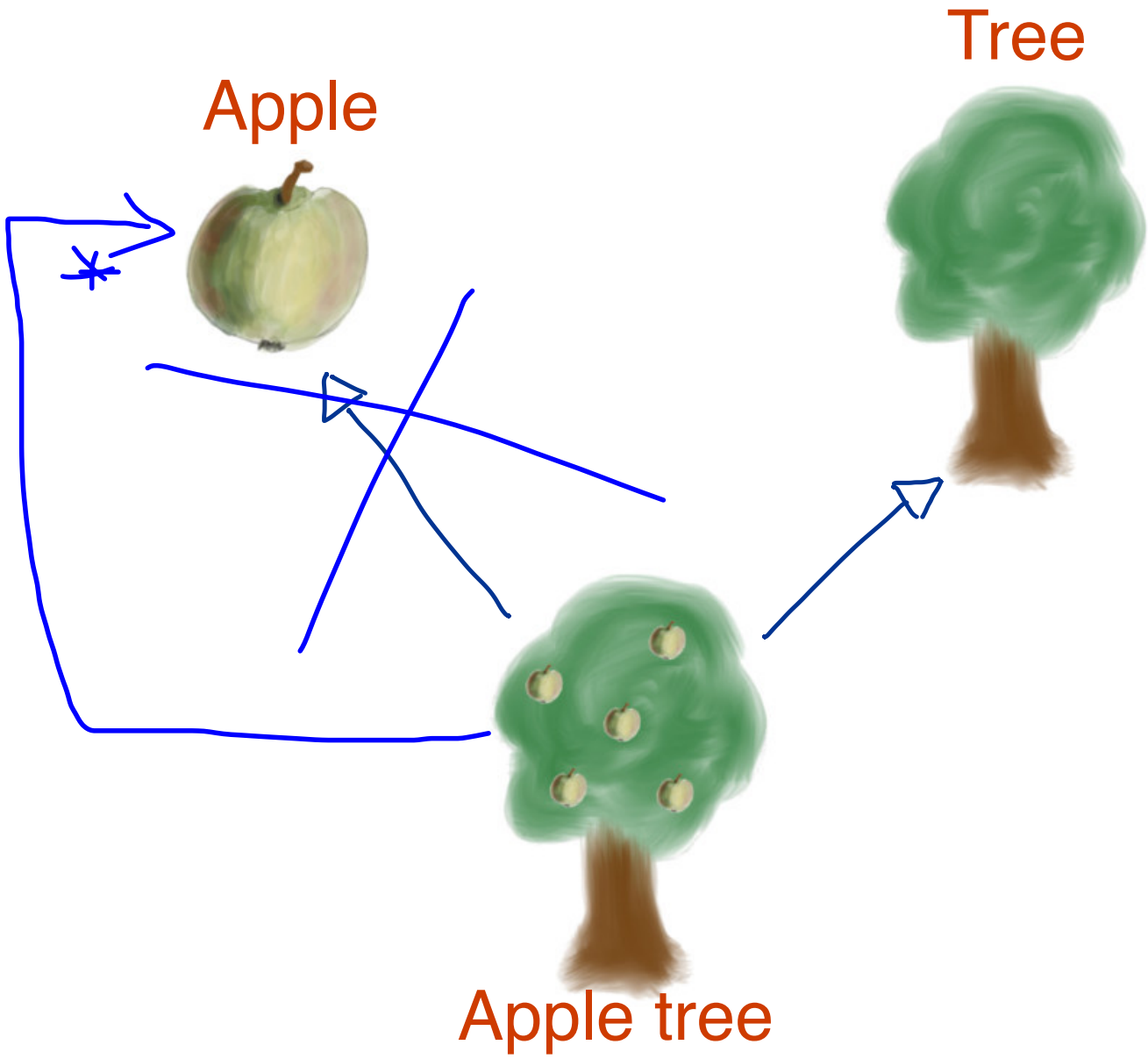
Generalisation Example



Liskov-Wing Substitution Principle

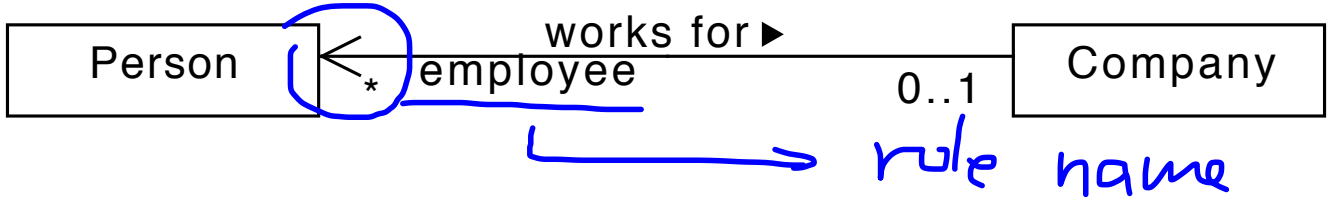
> *"If S is a subtype of T, then objects of type T in a program may be replaced with objects of type S without altering any of the desirable properties of that program (e.g., correctness)."*

Appletree



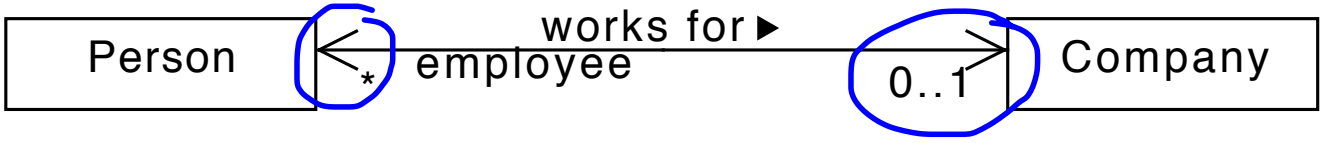
Associations between classes

- Unidirectional (association can be navigated in one direction)

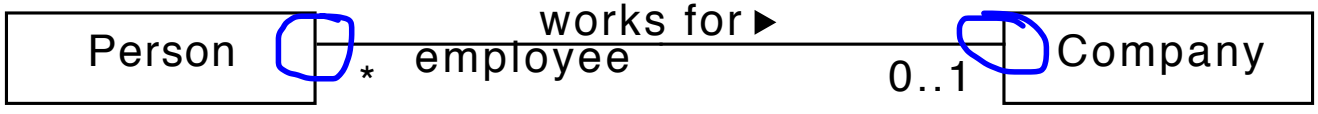


  - Company has a field employees

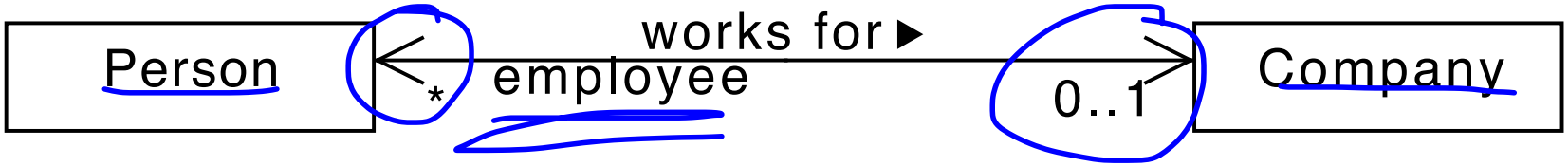
- Bidirectional (association can be navigated in both directions)
  - Company has a field employee*s* and a Person has a field company



- Bidirectional or no explicit navigability
  - no explicit navigability ≡ no fields



Associatons between classes




```
public class Person
{    ....
     private Company company;
     public getCompany() {
       return company;
     }
     public setCompany(Company c) {
       company = c;
     }
     ....
}
```

hire

fire

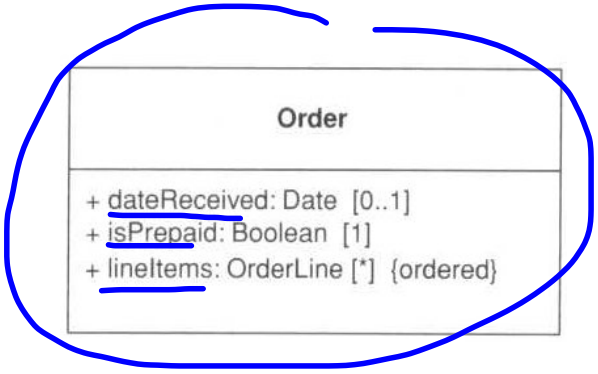
```
public class Company
{    ....
     private Set<Person> employees;
     ....
}
```

List<Person>

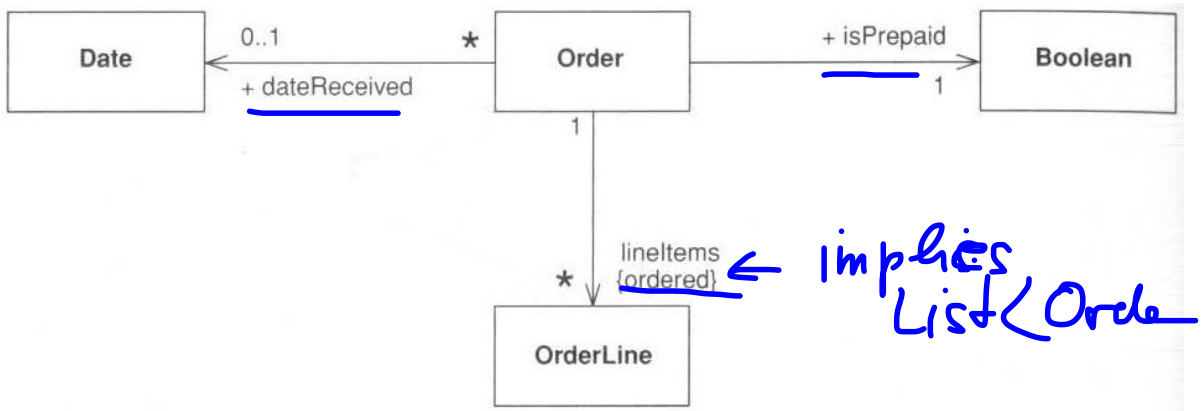
addEmployee

removeEmployee

Attributes and Associations




```
public class Order {
  private Date date;
  private boolean isPrepaid = false;
  private List<OrderLine> lineItems =
    new ArrayList<OrderLine)();
  ...
}
```



Contents

Sequence Diagram: Computing the price of an order

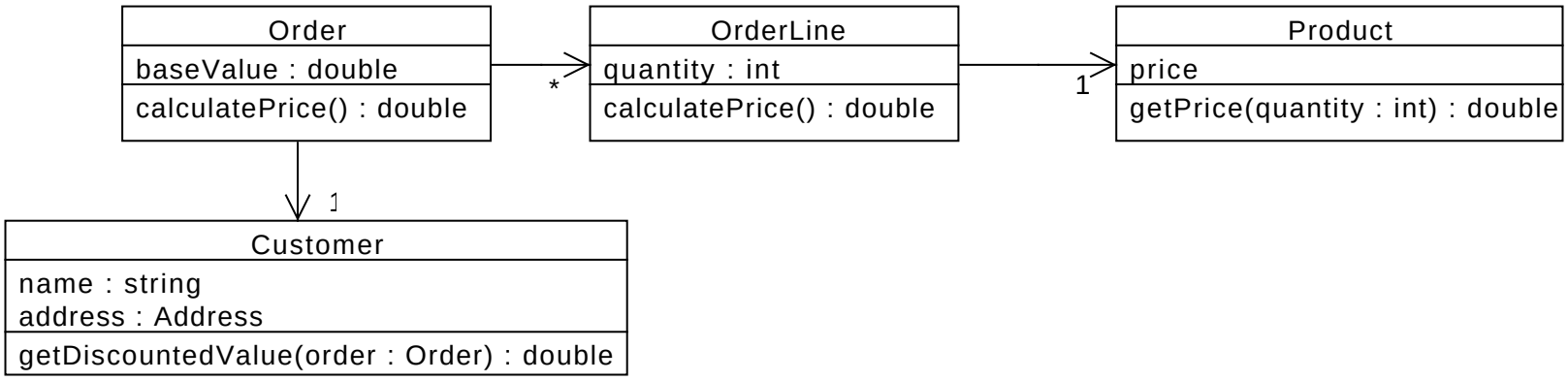
```
public class Order {

  List<OrderLine> orderLines = new ArrayList<OrderLine>();
  private Customer customer;
  double baseValue = 0;

  public double calculatePrice() {
    for (OrderLine ol : orderLines) {
      baseValue += ol.calculatePrice();
    }
    return customer.getDiscountedValue(this);
  }

  public double getBaseValue() {
    return baseValue;
  }
}
```

Sequence Diagram

```
public class OrderLine {

  public int quantity;
  public Product product;

  public double calculatePrice() {
    return product.getPrice(quantity);
  }

}


public class Product {

  public double price;

  public double getPrice(int quantity) {
    return price * quantity;
  }
}


public class Customer {

  public double getDiscountedValue(Order order) {
    return (1 - 5/100)*order.getBaseValue(); // 5% discount
  }
}
```

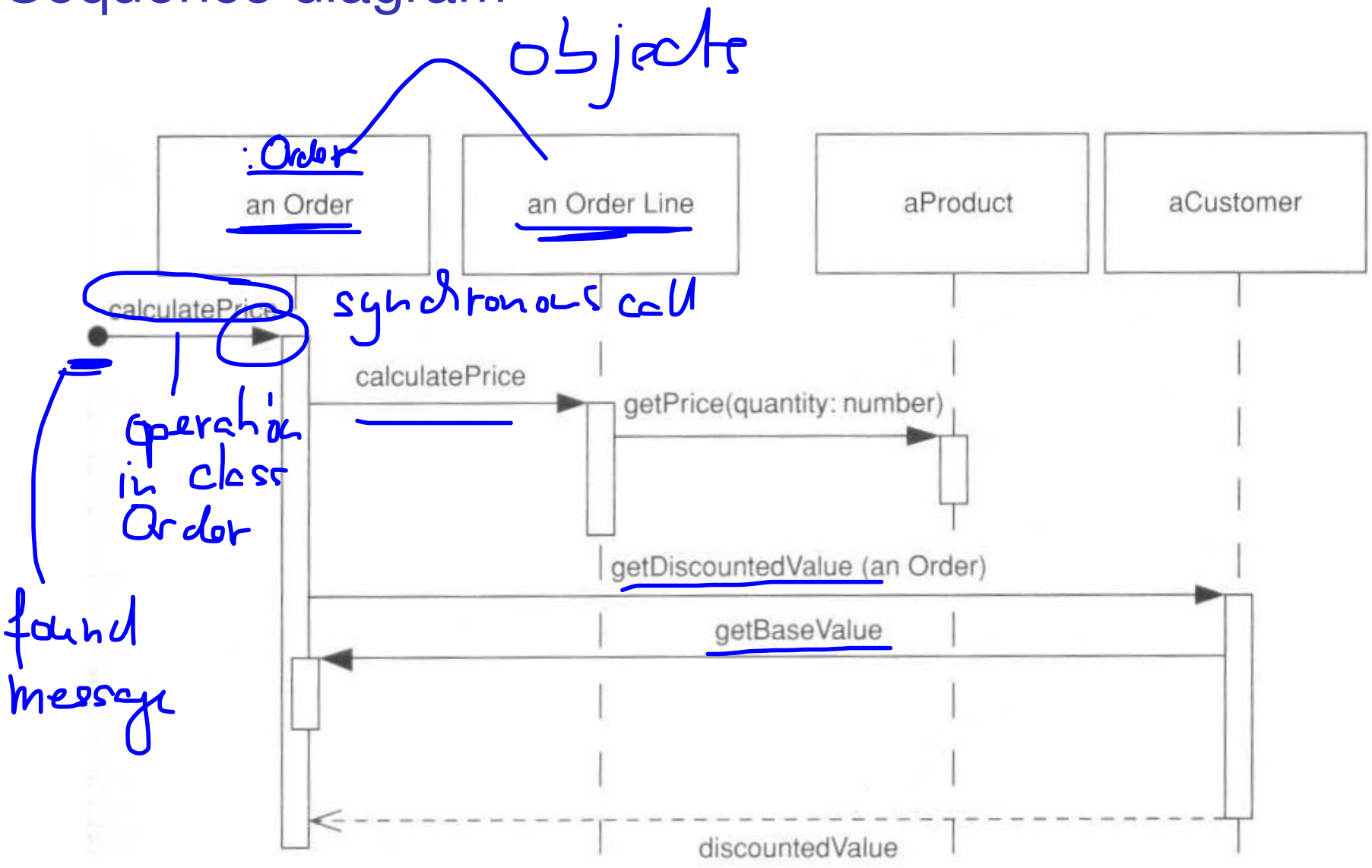
Sequence Diagram: Computing the price of an order

- Class diagram

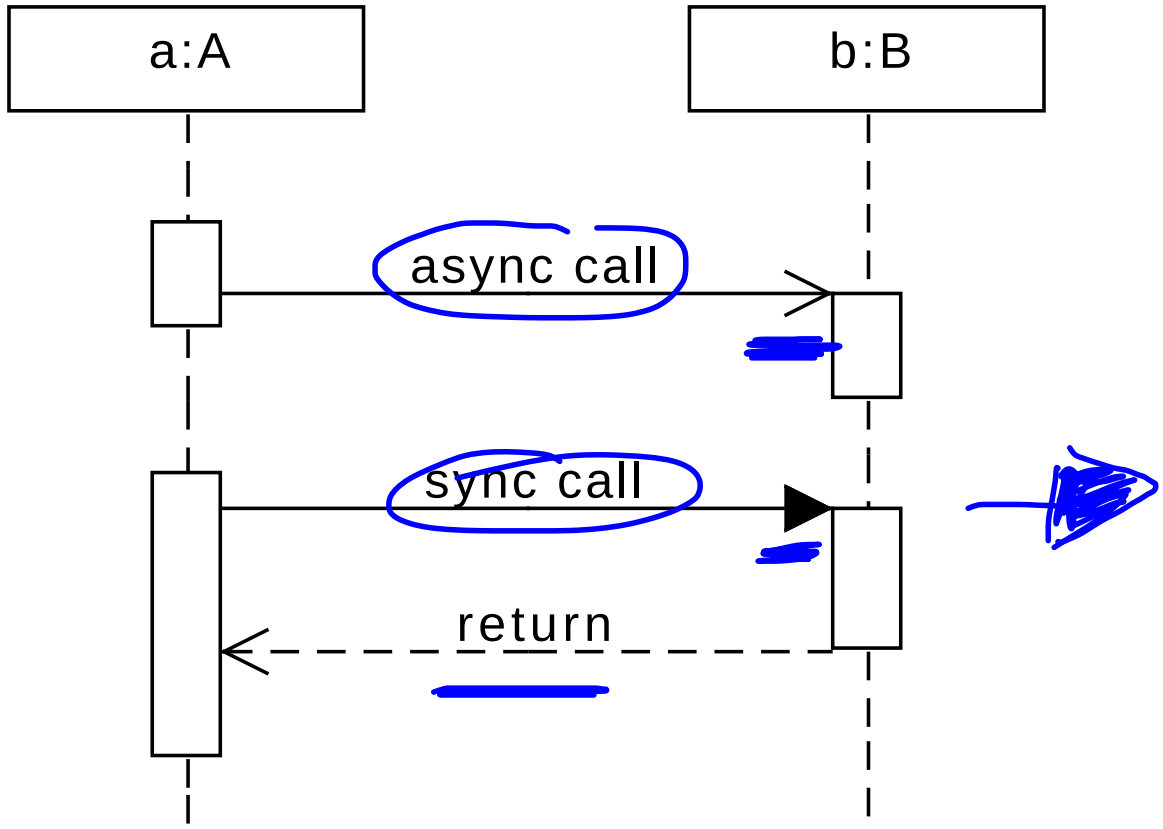


- Problem:
  - What are the operations doing?

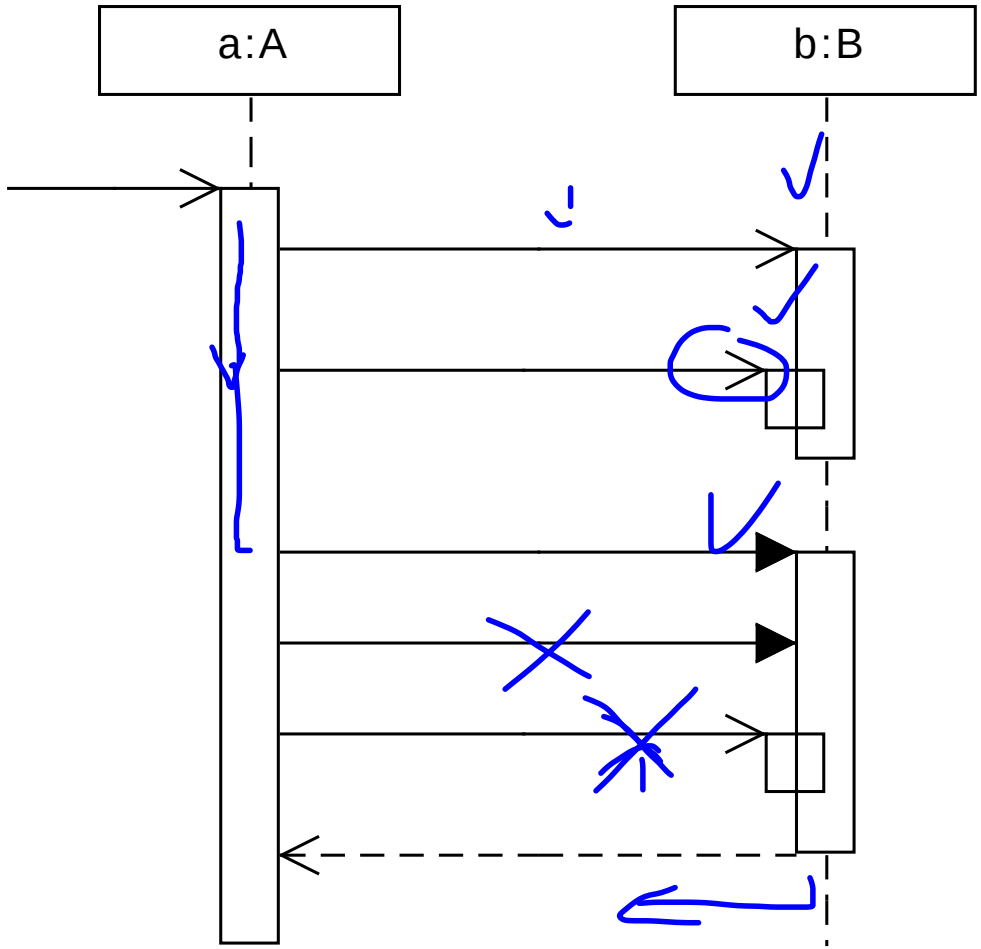
Sequence diagram



Arrow types



Arrow types



Usages of sequence diagrams

- Show the exchange of messages of a system
  - i.e. show the execution of the system
  - in general only *one* scenario
  - with the help of interaction frames also several scenarios
- For example use sequence diagrams for
  - Designing (c.f. CRC cards)
  - Visualizing program behaviour

Contents

Course 02161 Exam Project

- Week 5 (this week) + 6:
  - Requirements: Glossary, use case diagram, detailed use cases for selected use cases
  - Models: Class diagram plus sequence diagrams for previously selected detailed use cases
- Week 7:
  - Peer review of models from other groups
- Week 8—13:
  - Implementation
- Week 13:
  - 10 min demonstrations of the software are planned for Monday
    - → The tests need to be demonstrated

Introduction to the project

- What is the problem?
  - Project planning and time recording system
  - More information on CampusNet
- Create
  - Requirement specification
  - Programdesign
  - Implementation
  - Tests
- Deliver
  - Week 7: report describing the requirement specification and design
  - Week 8: feedback on the requirements and design of one other group
  - Week 13:
    - report on the implementation
    - **Eclipse** project containing the source code, the tests, and the running program (uploaded to CampusNet as a **ZIP** file that can be imported in Eclipse)
    - demonstration in front of TA's

Organisational issues

- Group size: 2 – 4
- Report can be written in Danish or English
- Program written in Java and tests use JUnit
- Each section, diagram, etc. should name the author who made the section, diagram, etc.
- **You can talk with other groups (or previous students that have taken the course) on the assignment, but *it is not allowed to copy from others parts of the report or the program*.**
  - *Any text copy without naming the sources is viewed as cheating*
- In case of questions with the project description send email to `huba@dtu.dk`

Week 5–6: Requirements and Design

Design process

1. Create glossary and use cases based on noun, adjectives, and verbs in the user requirements document
2. Create user stories based on use case scenarios
3. Create a set of initial classes based on nouns from the glossary $\rightarrow$ initial design
3. Take one user story
   a) Design the system by executing the user story in your head
      $\rightarrow$ e.g. using CRC cards
   b) Extend the existing class diagram with classes, attributes, and methods
3. Repeat step 2 with the user stories

Week 7: Peer Review the models of your colleagues

Criteria to check for

- Use of the correct notation (use case diagram, class diagram, sequence diagrams)
- Consistency and completeness of the models
  - use case names in use case diagrams are the same as use case names in detailed use cases
  - sequence diagrams fit to the user stories
  - use case diagram describes the complete behaviour of the system
  - ...
- Readability
  - Do you understand the model?

Learning objectives of Week 4—7

- Learn to think abstractly about object-oriented programs
  - Using programming language independent concepts
- Learn how to communicate requirements and design
  - Requirements are read by the customer but also by the programmers
  - Have a language to talk with fellow programmers about design issues (class and sequence diagrams)
- By commenting on the models of others, you learn how people will read and understand your model
- I don't expect you to create perfect models
  - It is okay if your final implementation does not match your model
    - Focus on getting the notation and concepts right
  - By comparing your model with your final implementation, you learn about the relationship between modelling and programming

Week 8—13

Implementation process

1. Choose a set of user stories to implement
1. Select the user story with the highest priority
   a) Create the acceptance test for the story in JUnit
   b) Implment the user story test-driven, creating additional tests as necessary and **guided** by your desing
      → based on the classes, attributes, and methods of the model
      → implement **only** the classes, attributes, and methods needed to implement the user story
      → Criteria: 100% code coverage based on the tests you have
3. Repeat step 2 with the user story with the next highest priority

Grading

- The project will be graded as a whole

→ no separate grades for the model, the peer review, and the implementation