Software Engineering I (02161)

Week 6: Design 1: CRC cards, class– and sequence diagram

Assoc. Prof. Hubert Baumeister

DTU Compute
Technical University of Denmark

Spring 2017

Contents

Midterm evaluation

- Majority has decided to keep the time of the lecture from 15:00–17:00 (45% keep / 33% change / 24% okay with both)
- Course focuses on Java and object-oriented software
- Non-programming homework intended to be done **after** the lecture at home and not before
- Assignments
  - Programming exercises: not mandatory latest DL for feedback 19.3
  - Non-programming exercises: not mandatory latest DL for feedback 19.3
  - Examination project: mandatory assignments week 8 and week 13
- Need your help
  - How can I make the lecture more exciting?
  - How can I improve the Web site?

Contents

Recap

- week 1–3: Requirements
- week 3-5: Tests
  - week 5: Systematic tests and code coverage
- week 6-8: Design
- week $>$8: Implementation

Contents

From Requirements to Design

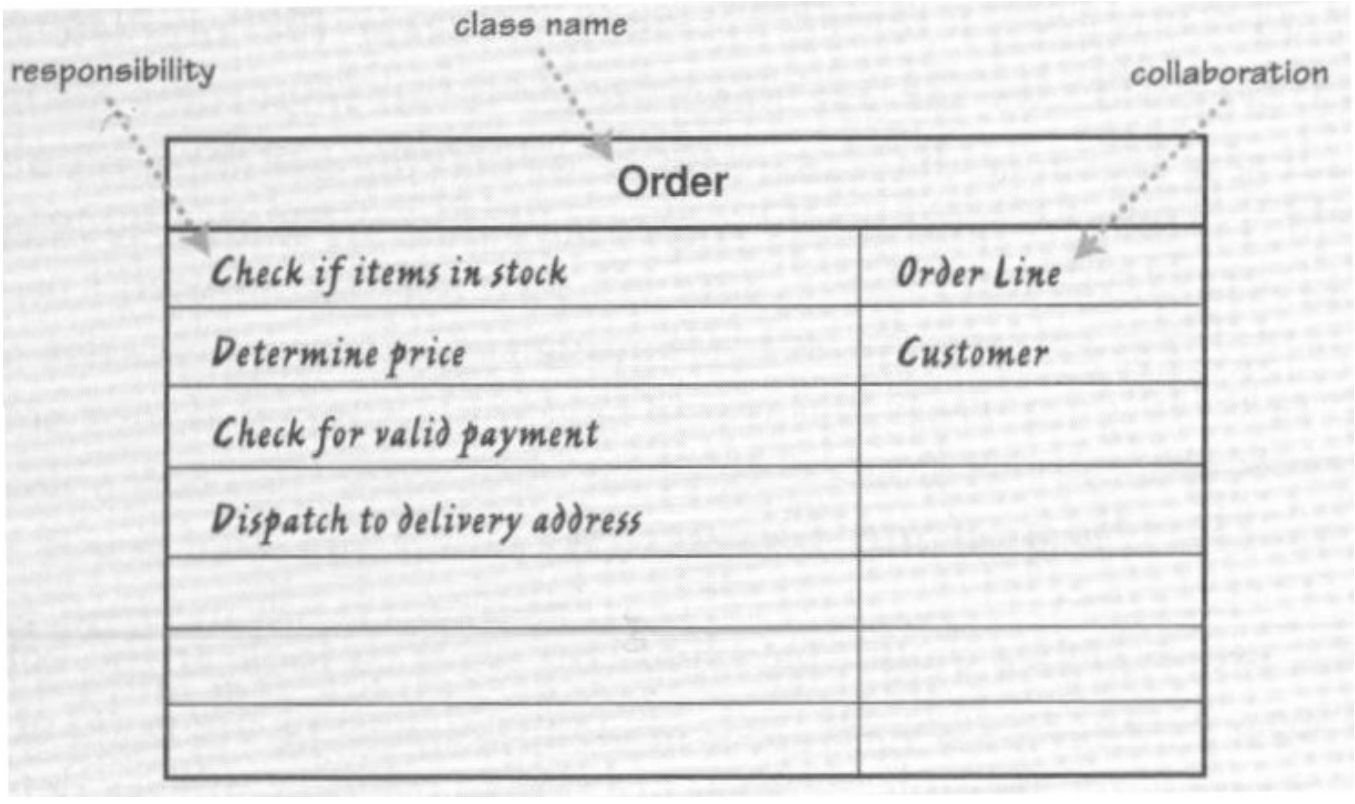
Design process (abstract)

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
   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

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



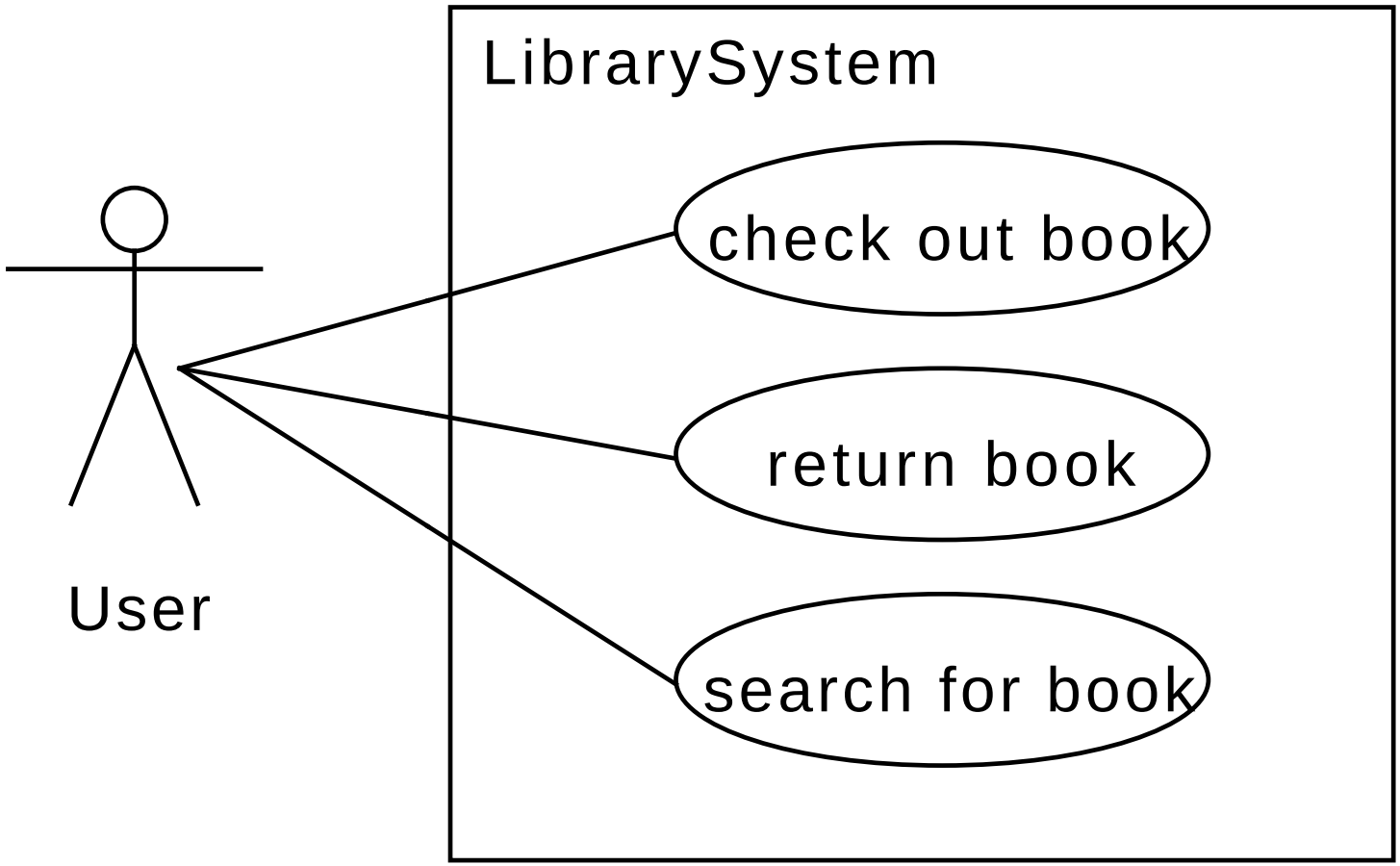
A larger example

- `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



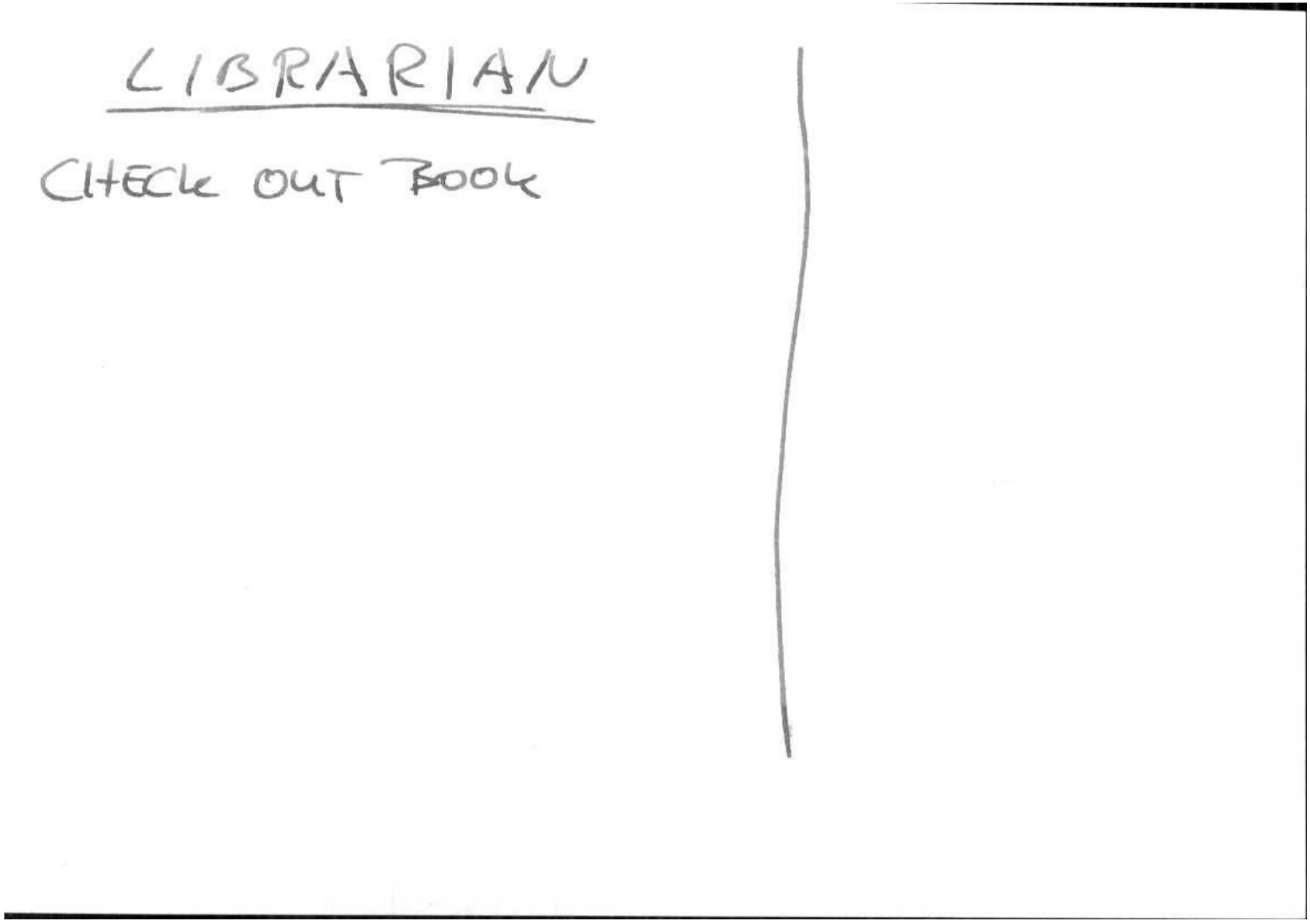
Library Example: Detailed Use Case *Check Out Book*

- **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

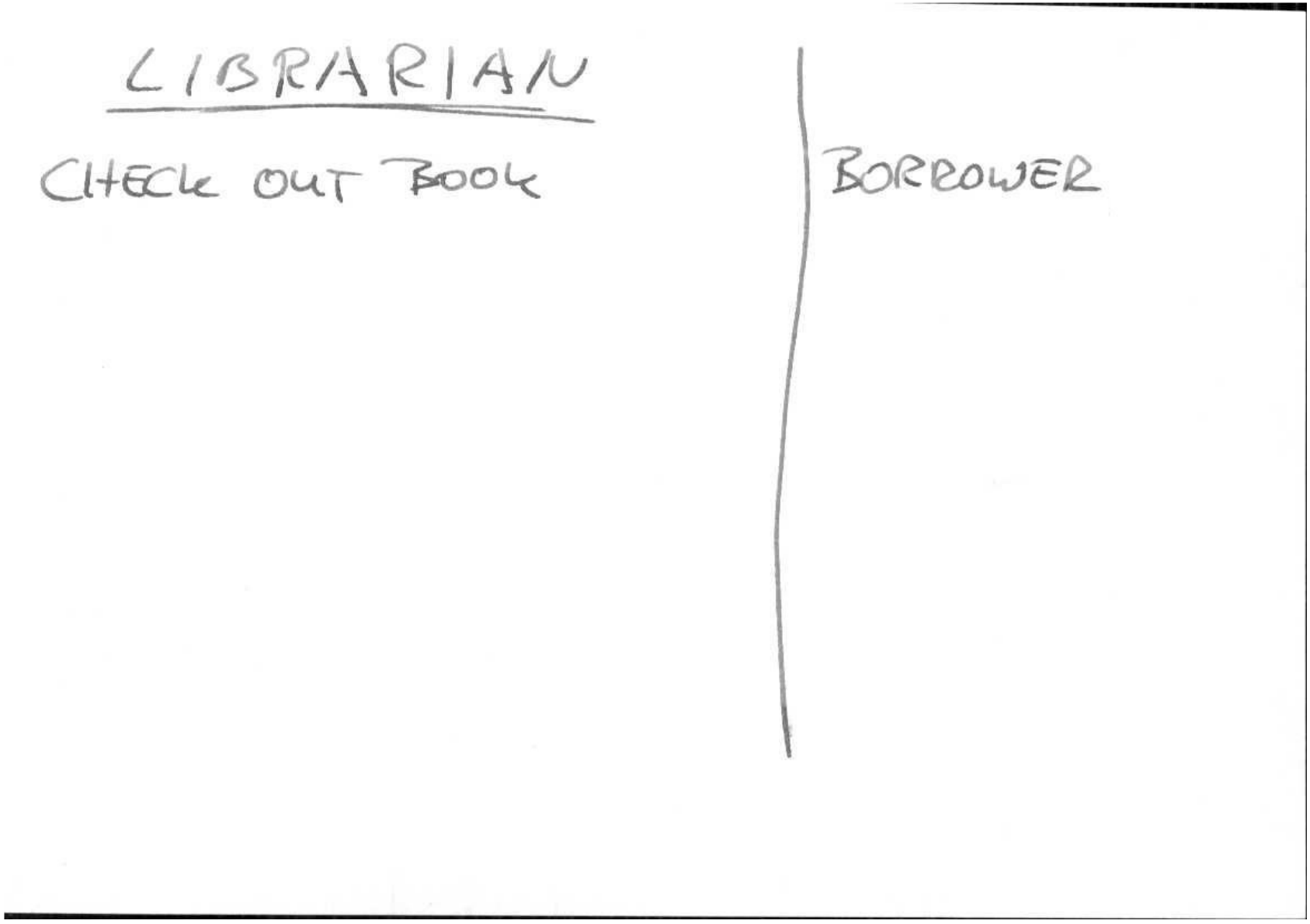
Example II

- 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+?”

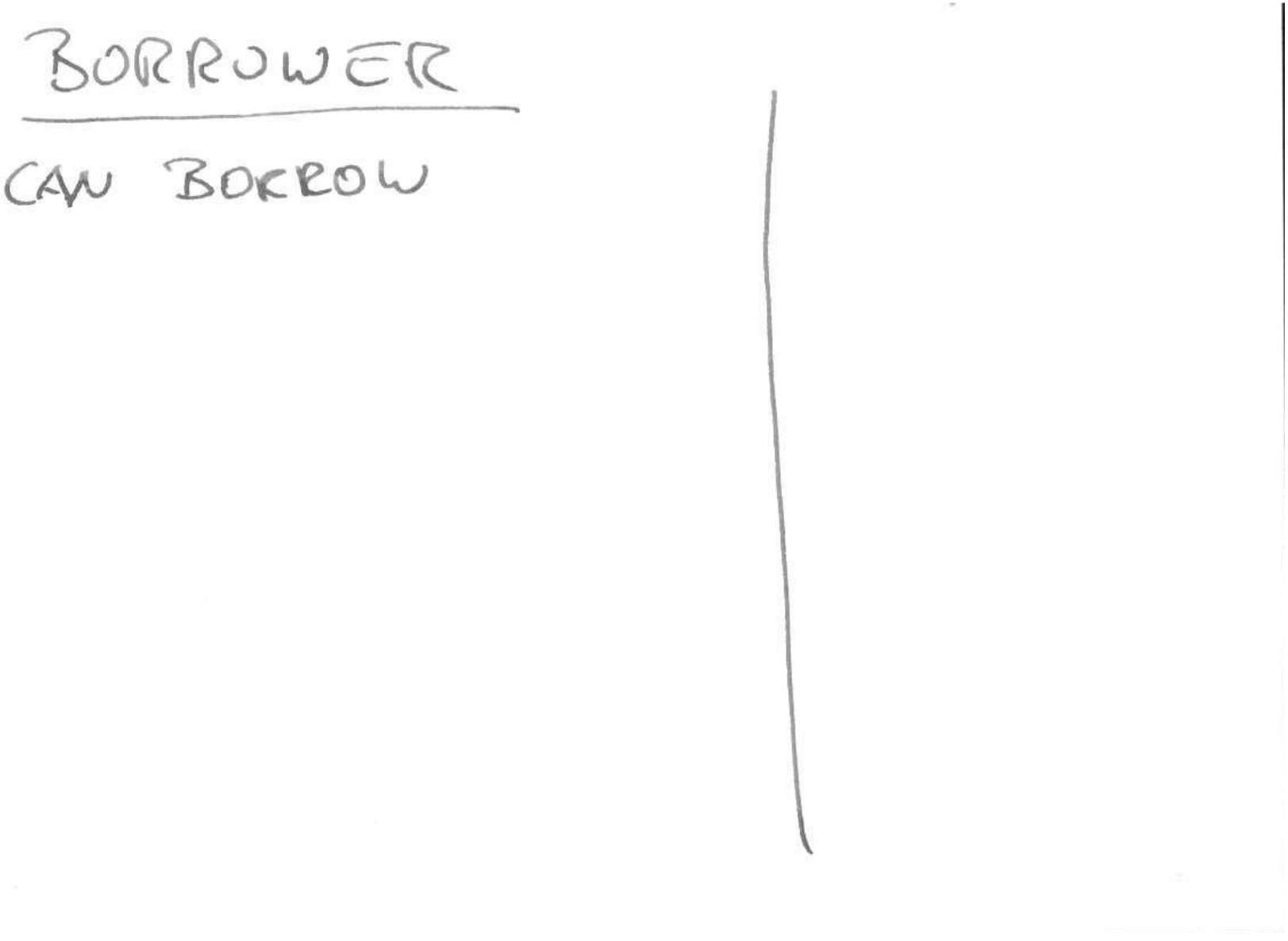
Library Example: CRC cards



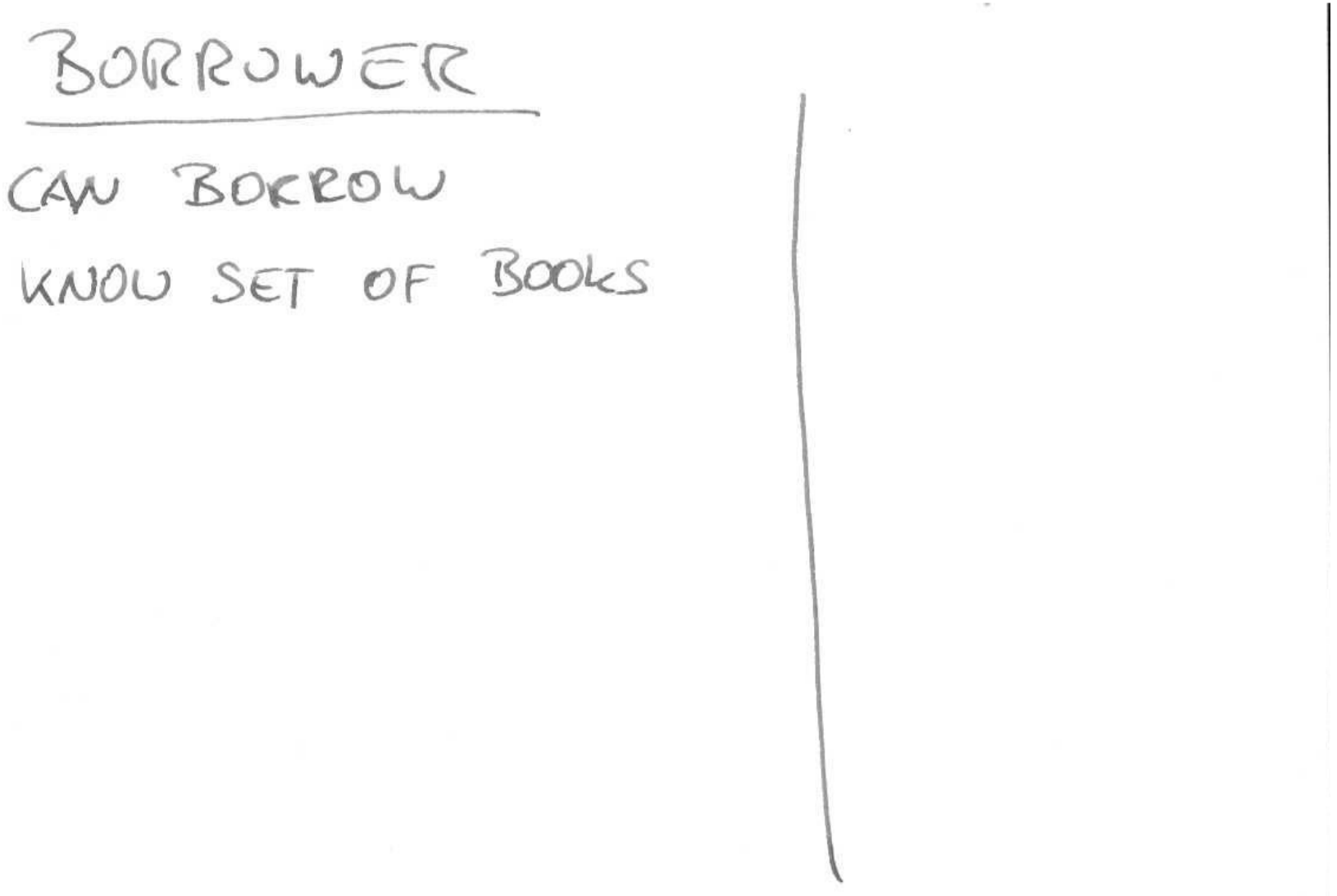
Library Example: CRC cards



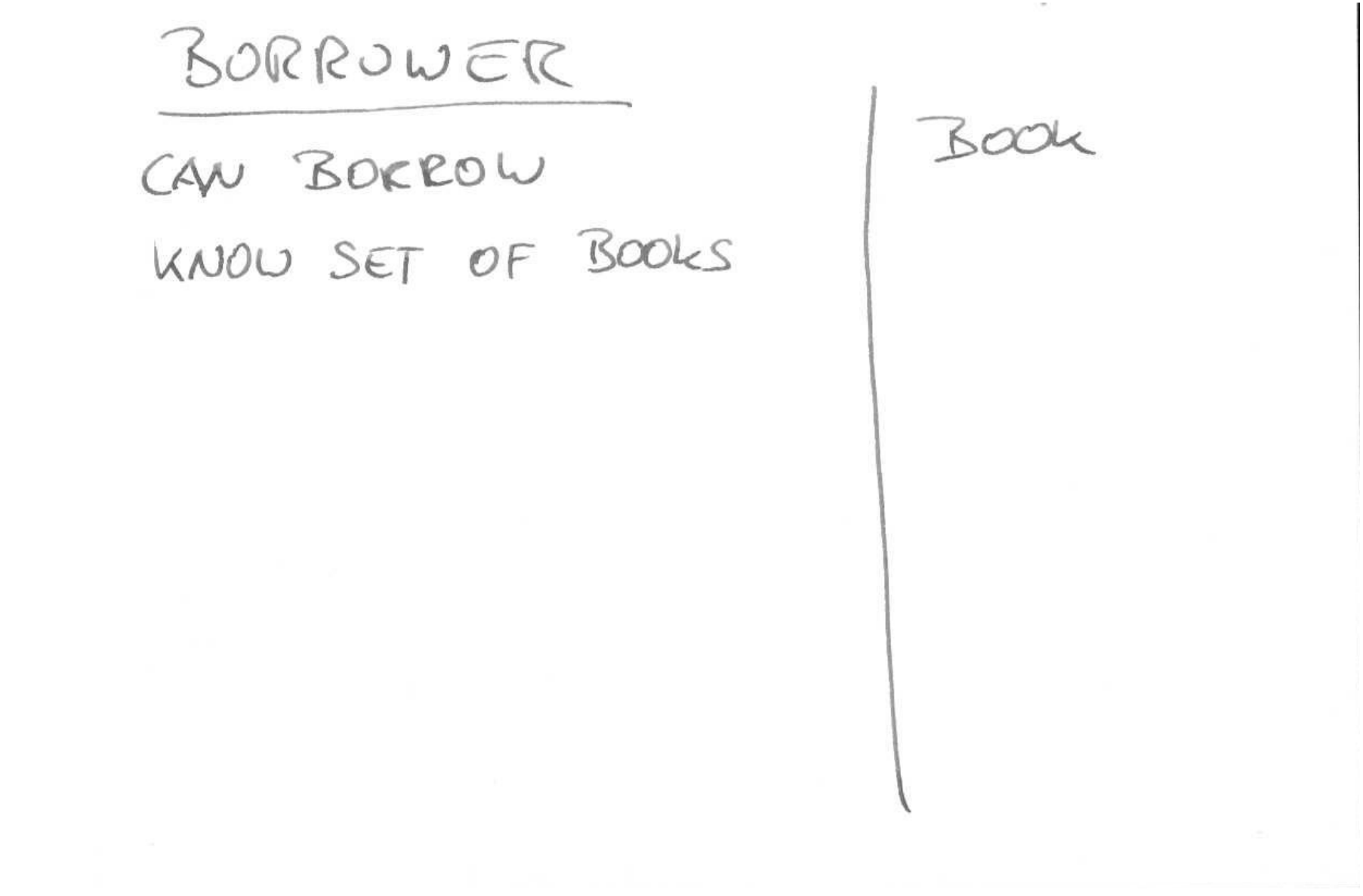
Library Example: CRC cards



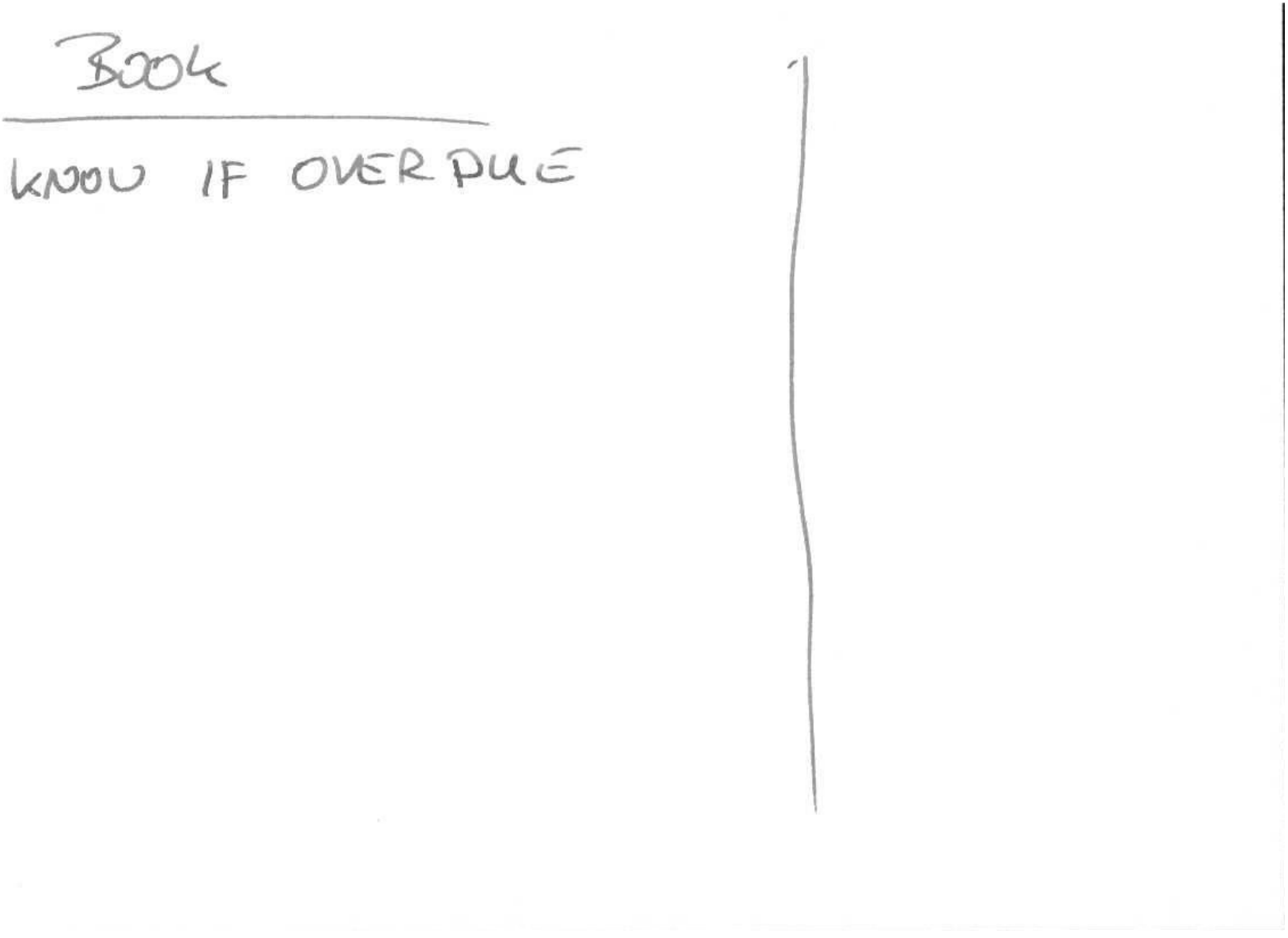
Library Example: CRC cards



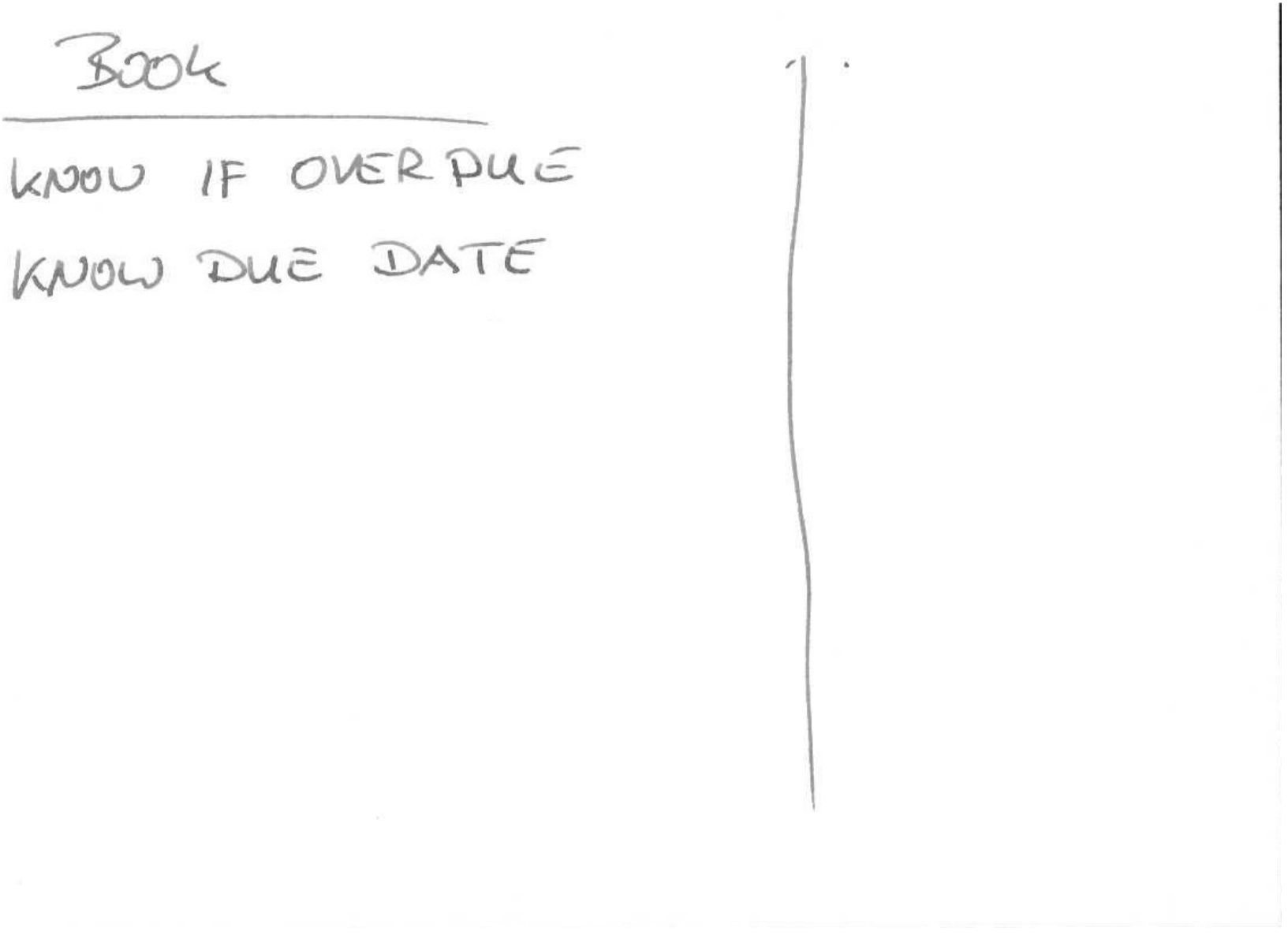
Library Example: CRC cards



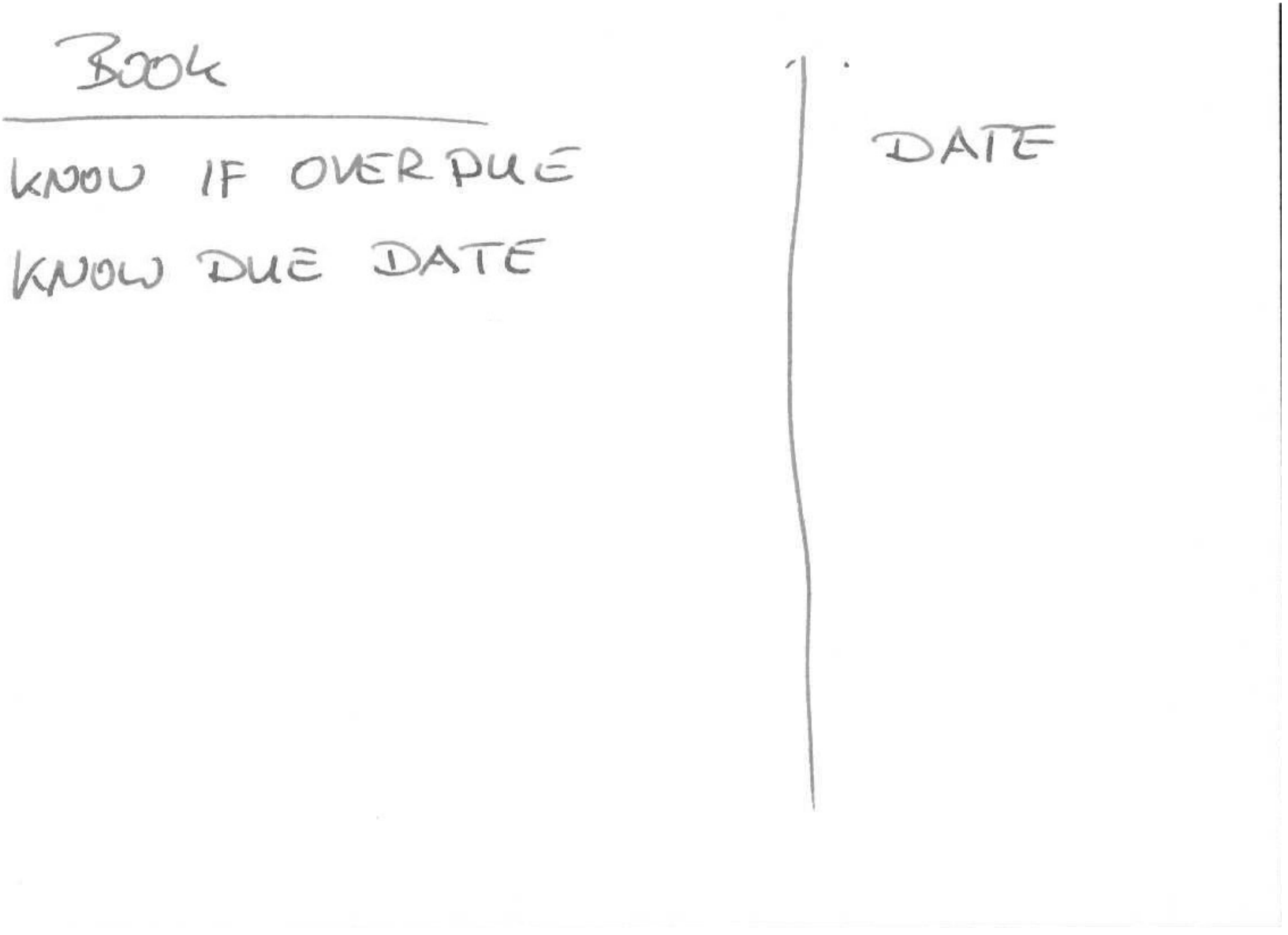
Library Example: CRC cards



Library Example: CRC cards

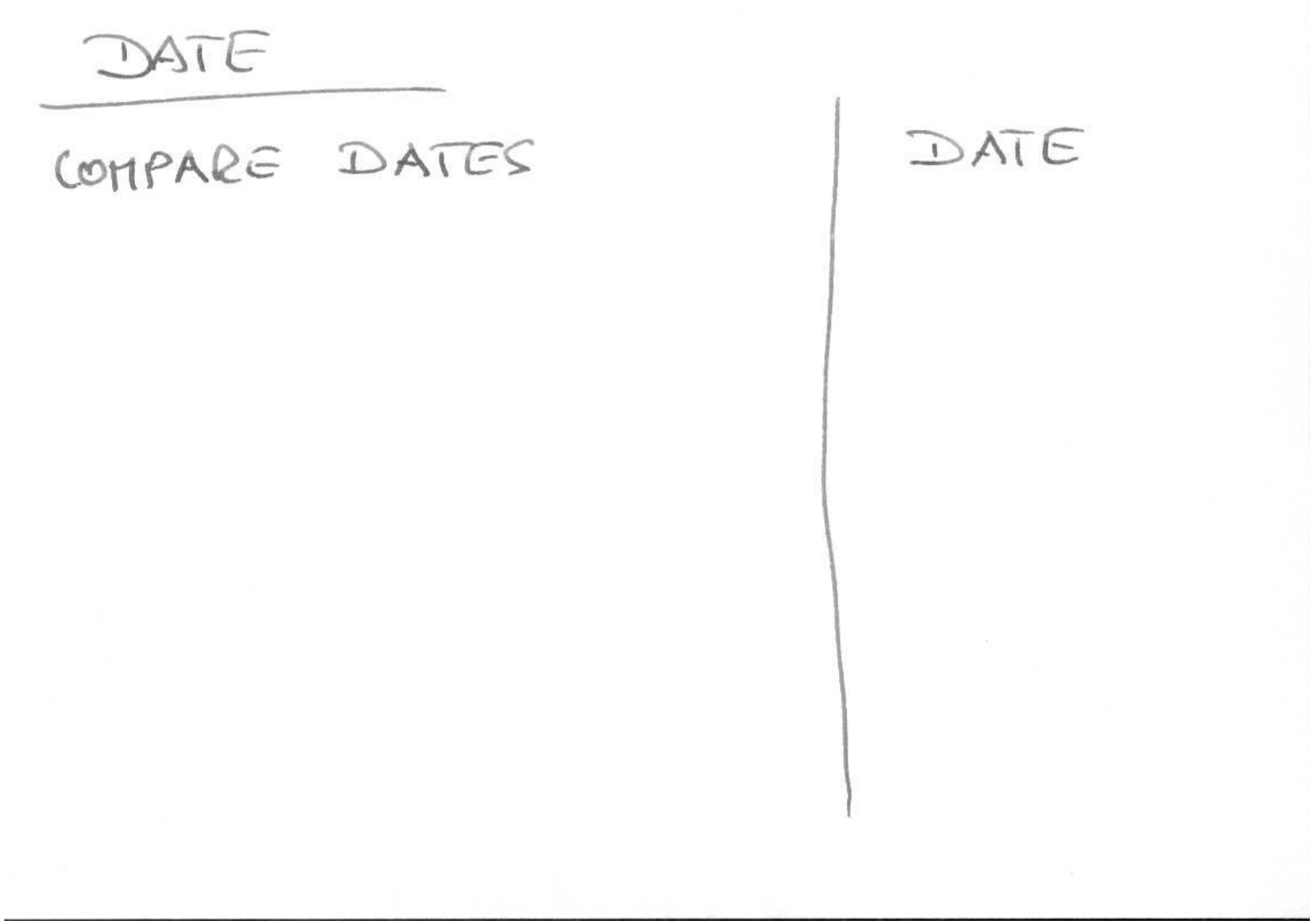


Library Example: CRC cards

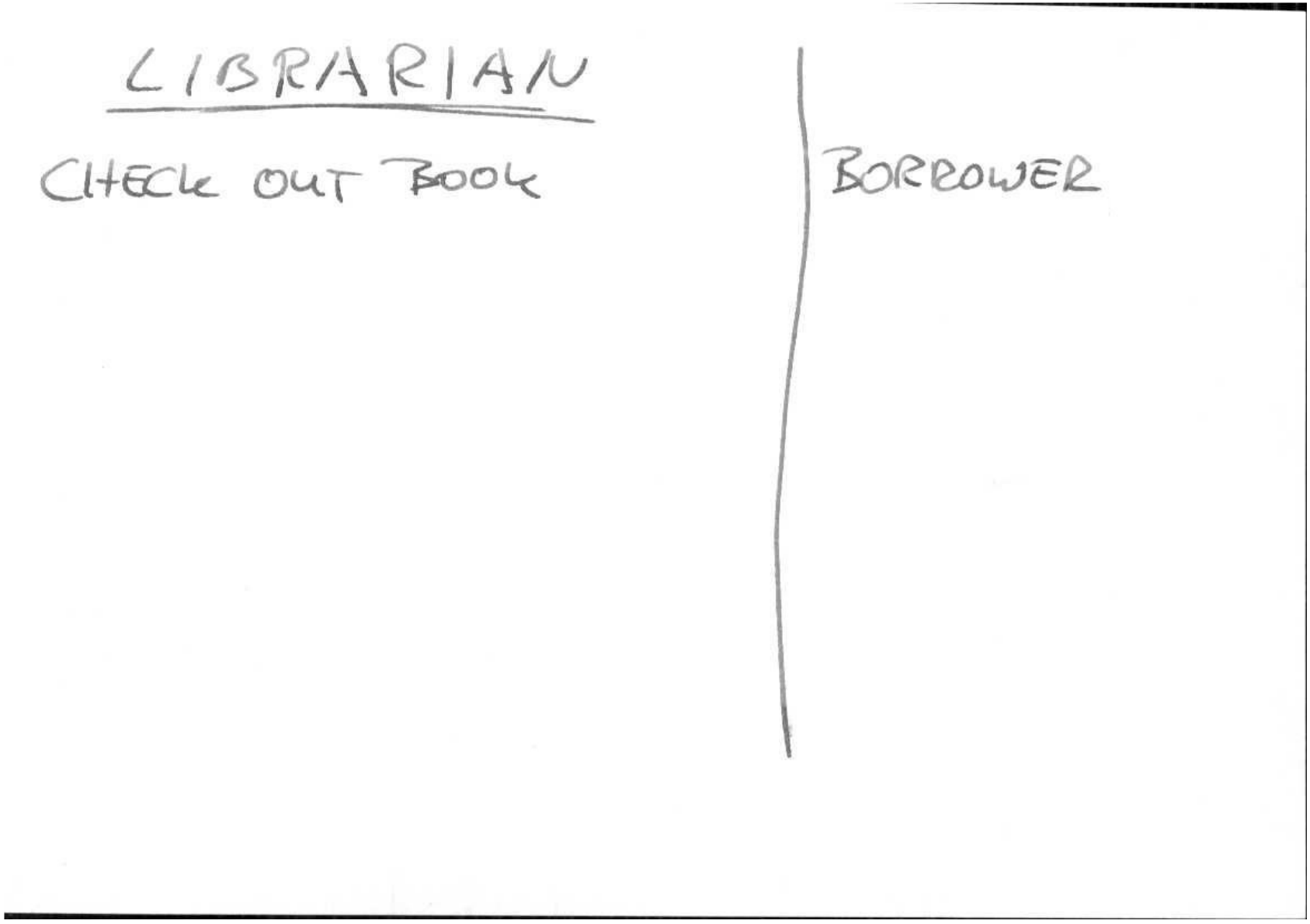


Library Example: CRC cards

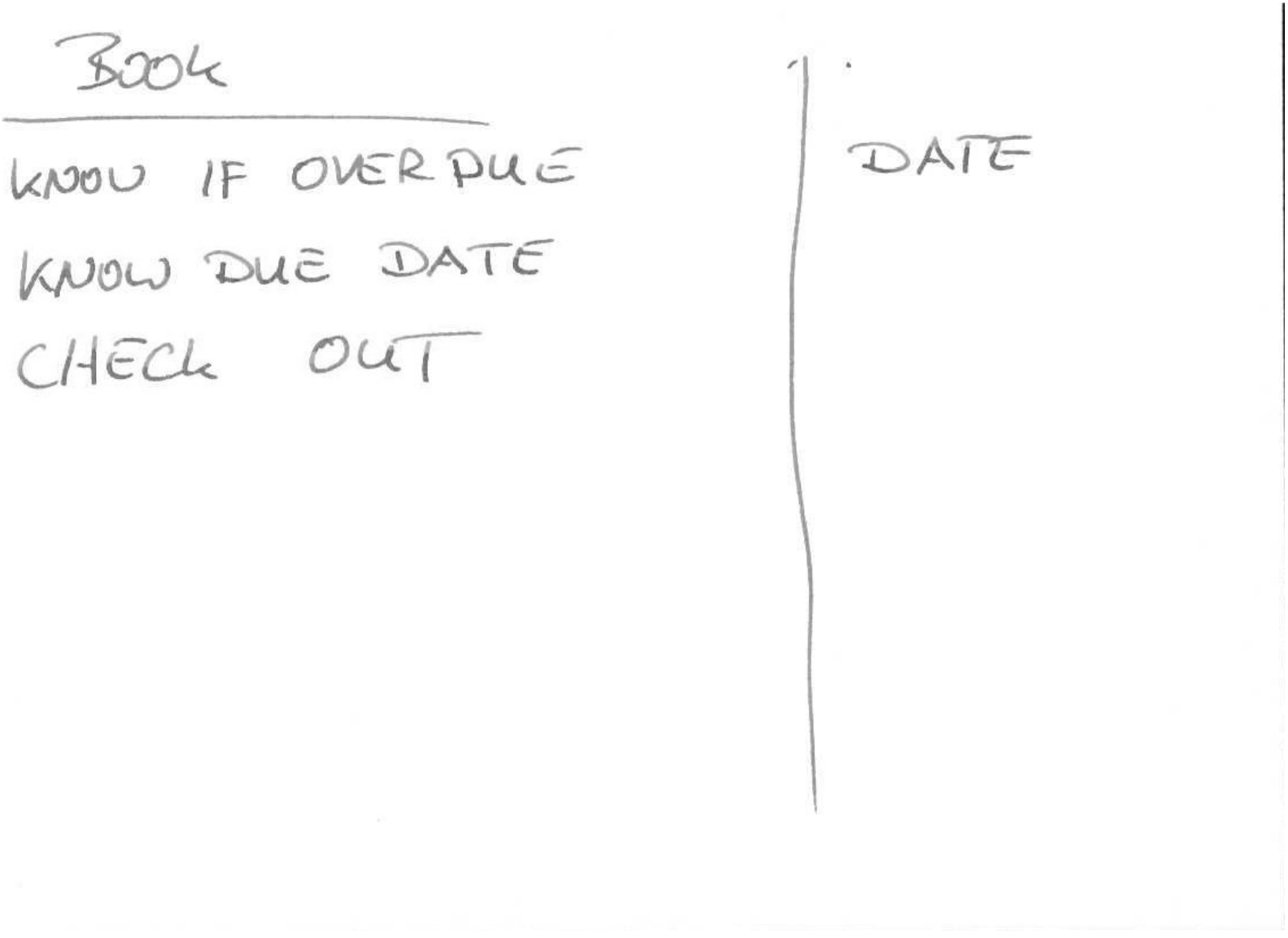
Library Example: CRC cards



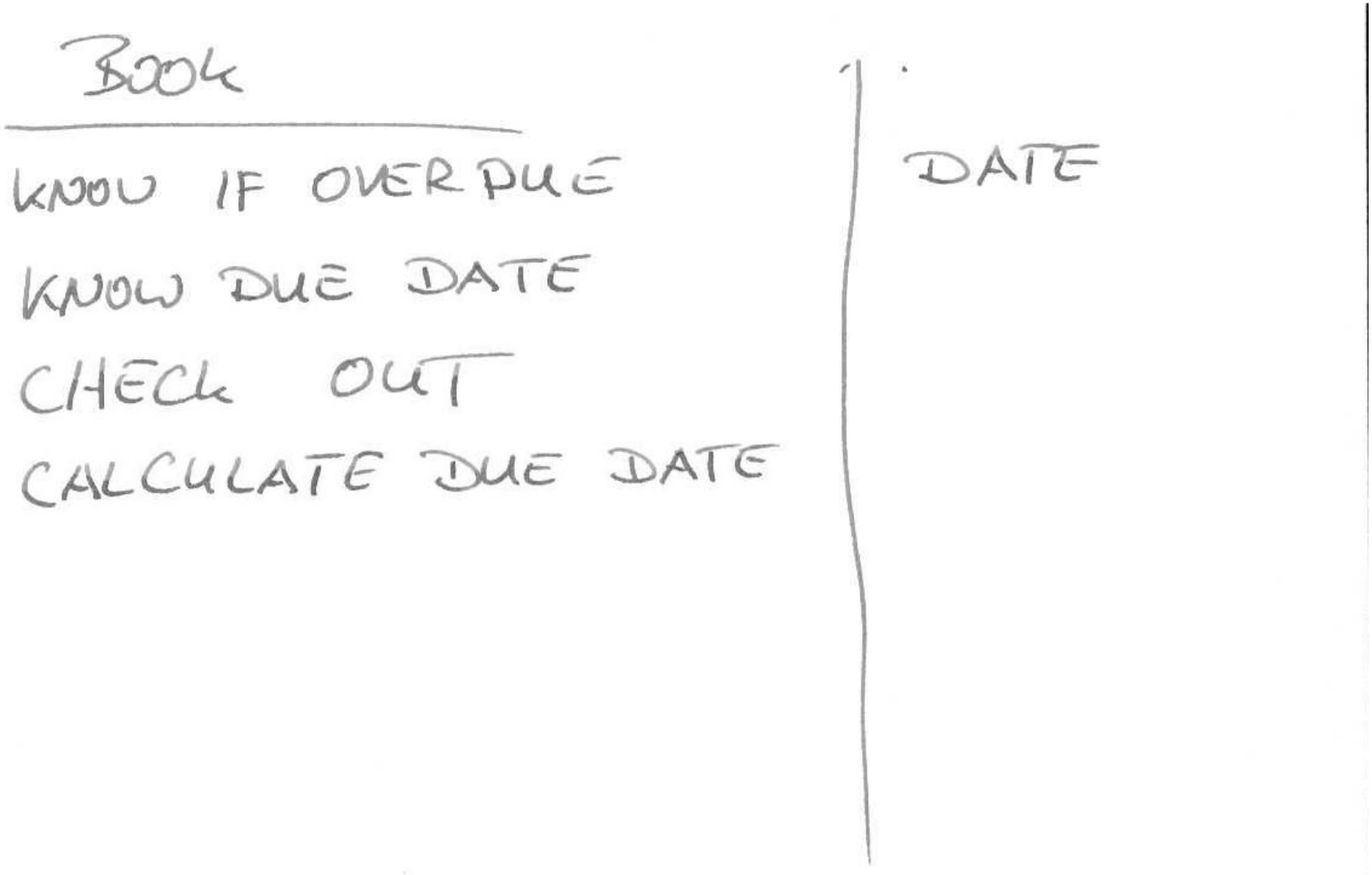
Library Example: CRC cards



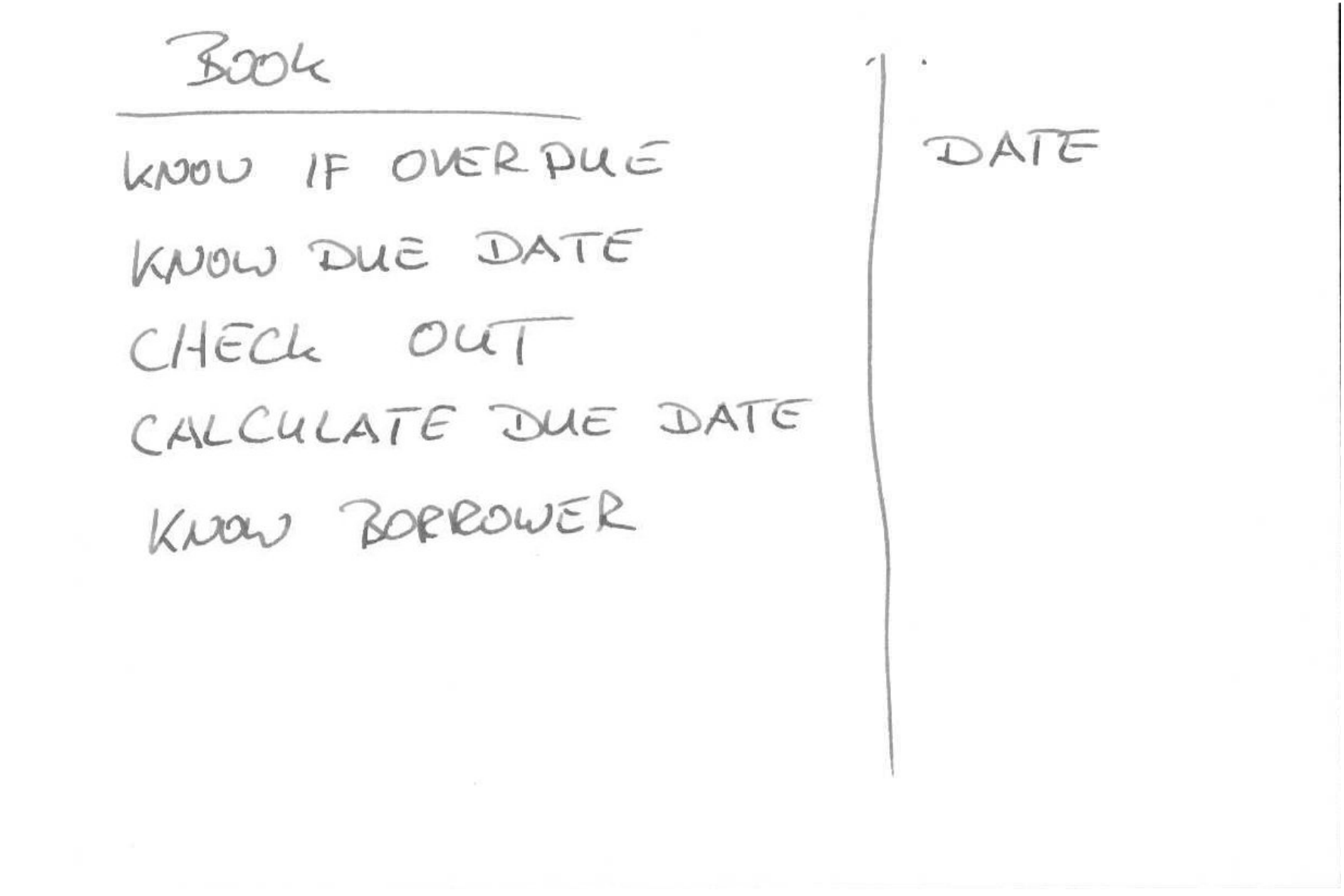
Library Example: CRC cards



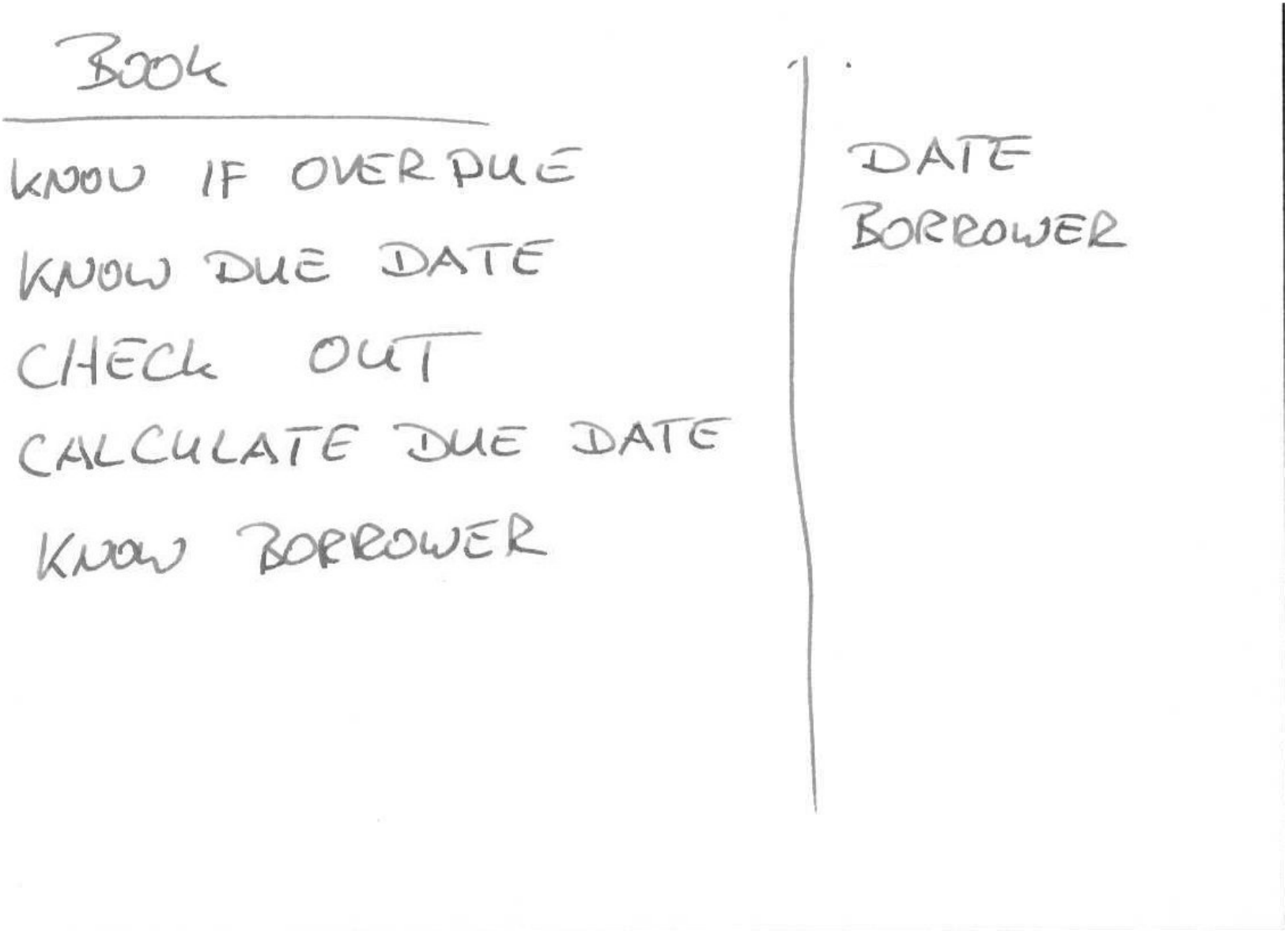
Library Example: CRC cards



Library Example: CRC cards



Library Example: CRC cards

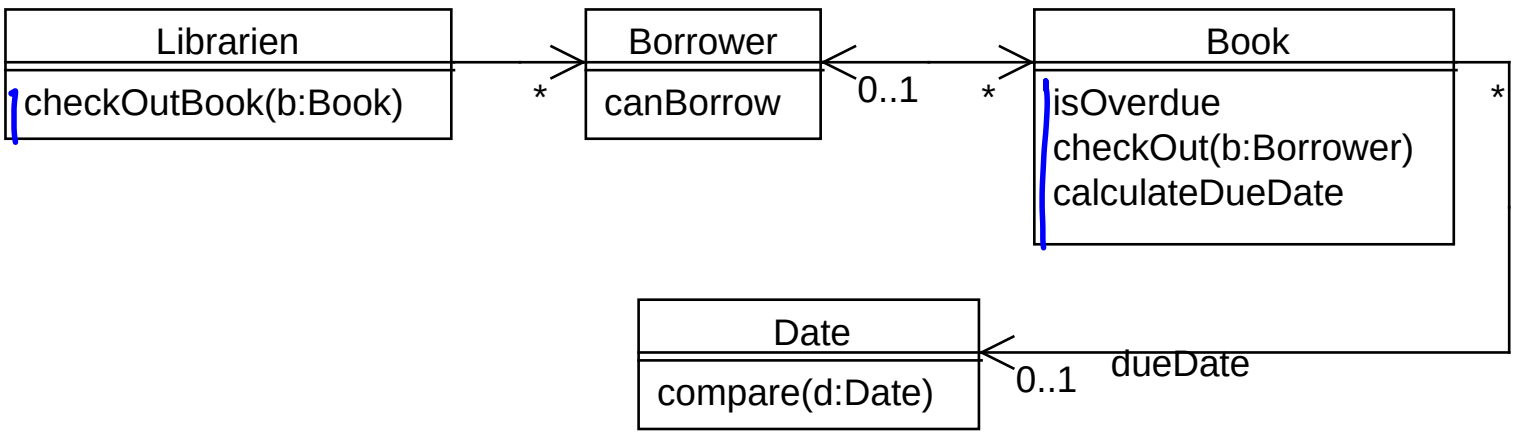


Library Example: All CRC cards

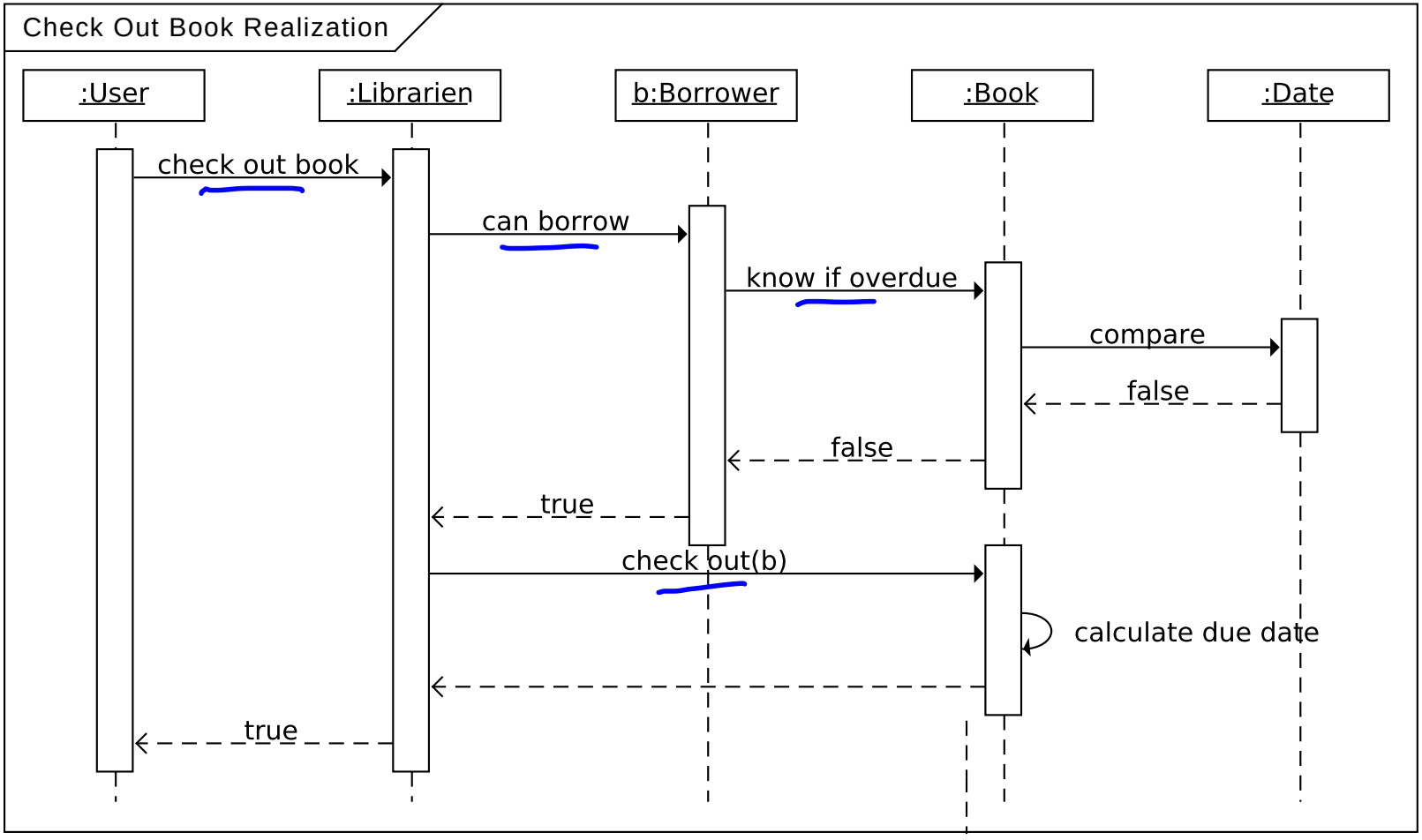
Process: Next Steps

- Review the result
  - Group cards
  - Check cards
  - Refactor
- Transfer the result
  - Either implement the user story based on the set of cards
  - Or create UML model documenting your design

Example: Class Diagram (so far)



Example: Sequence Diagram for Check-out book



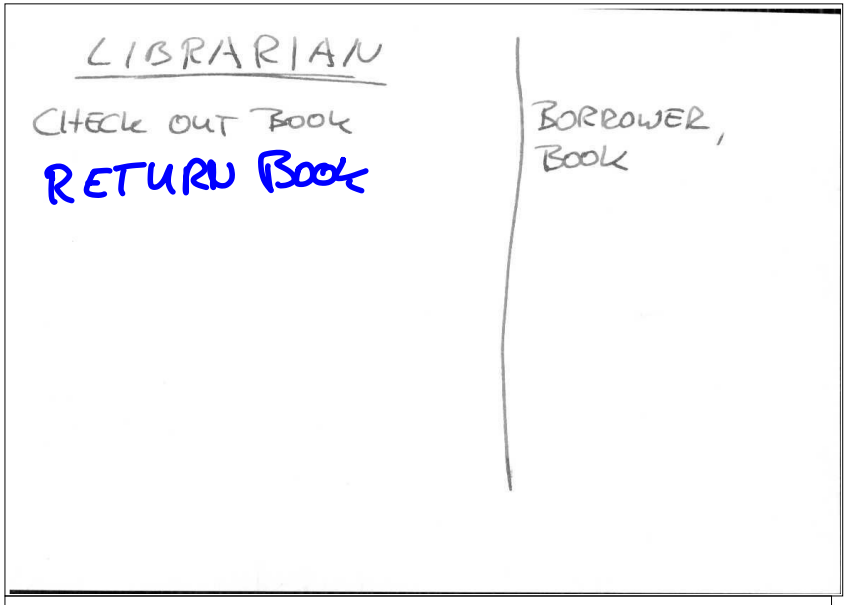
Alternative

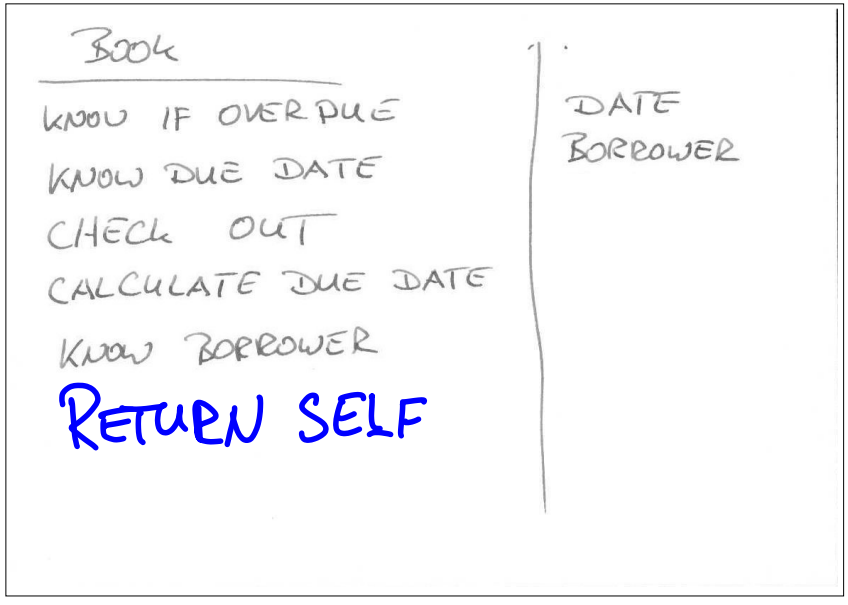
- Build class and sequence diagrams directly
  - Danger: talk about the system instead of being part of the system
  - Possible when object-oriented principles have been learned
  - CRC cards help with object-oriented thinking

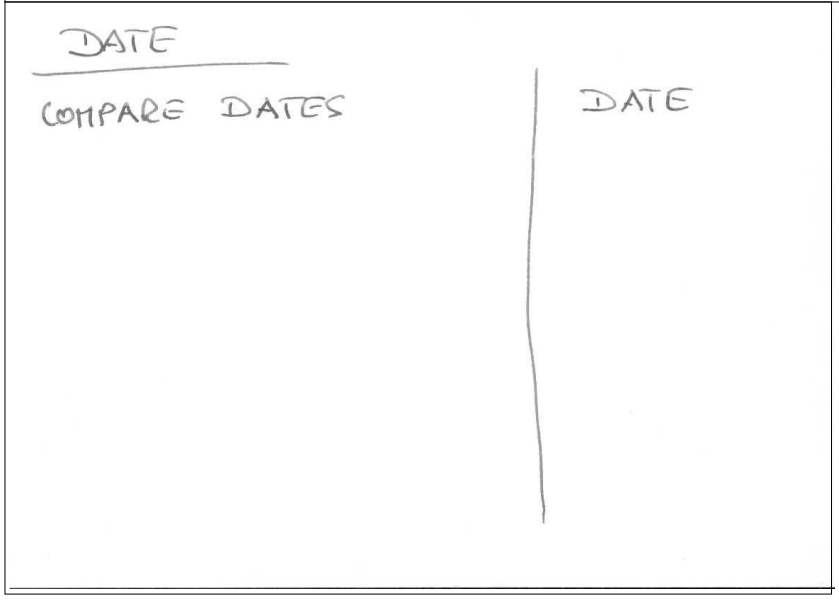
Exercise: Detailed Use Case *Return Book*

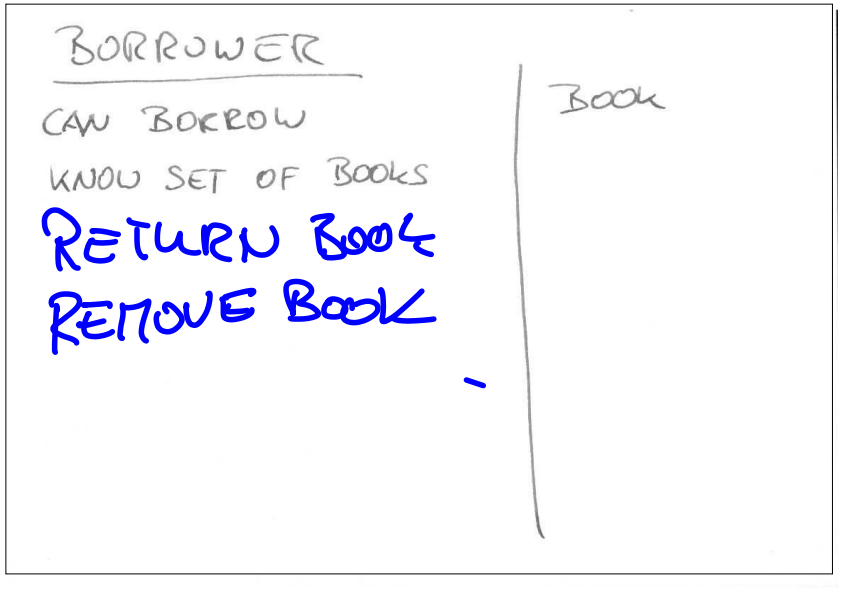
- **Name:** Return Book
- **Description:** The user retuns a book he had checked-out to the library
- **Actor:** User
- **Precondition** The book is checked-out by the user
- **Main scenario:**
  1. A user presents the book for check-in at the check-in counter
  2. The system registers that the book has been returned
- **Alternative scenarios:**
  - The book is overdue
    - 2a. The system calculates the fine and sends a bill to the customer
    - 2b. The system registers the return of the book

Exercise: Previous set of CRC cards









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

- Purpose:
  - Docmenting the domain
  - Documenting the design of a system
  - A language to talk about designs with other programmers

Why a graphical notation?

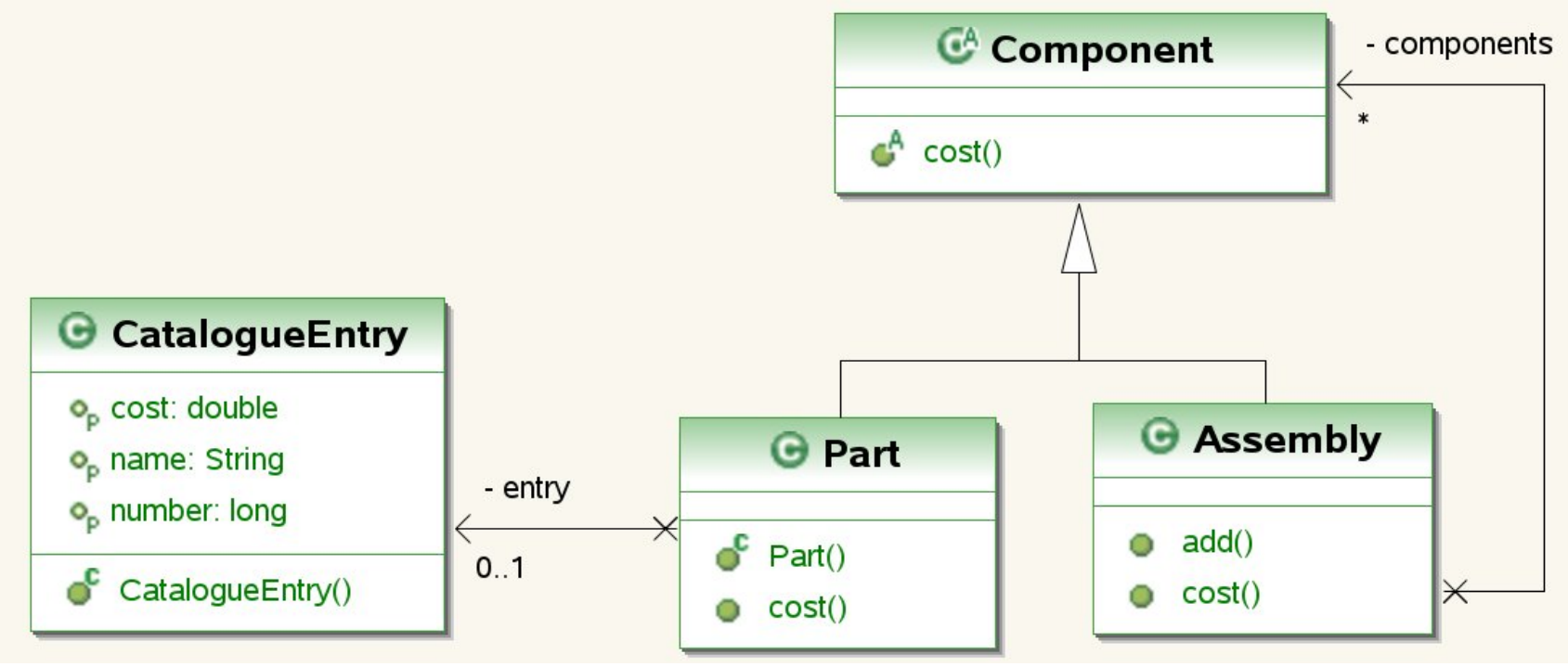
```
public class Assembly
            extends Component {
 public double cost() {   }
 public void add(Component c) {}
 private Collection<Component>
      components;
}

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

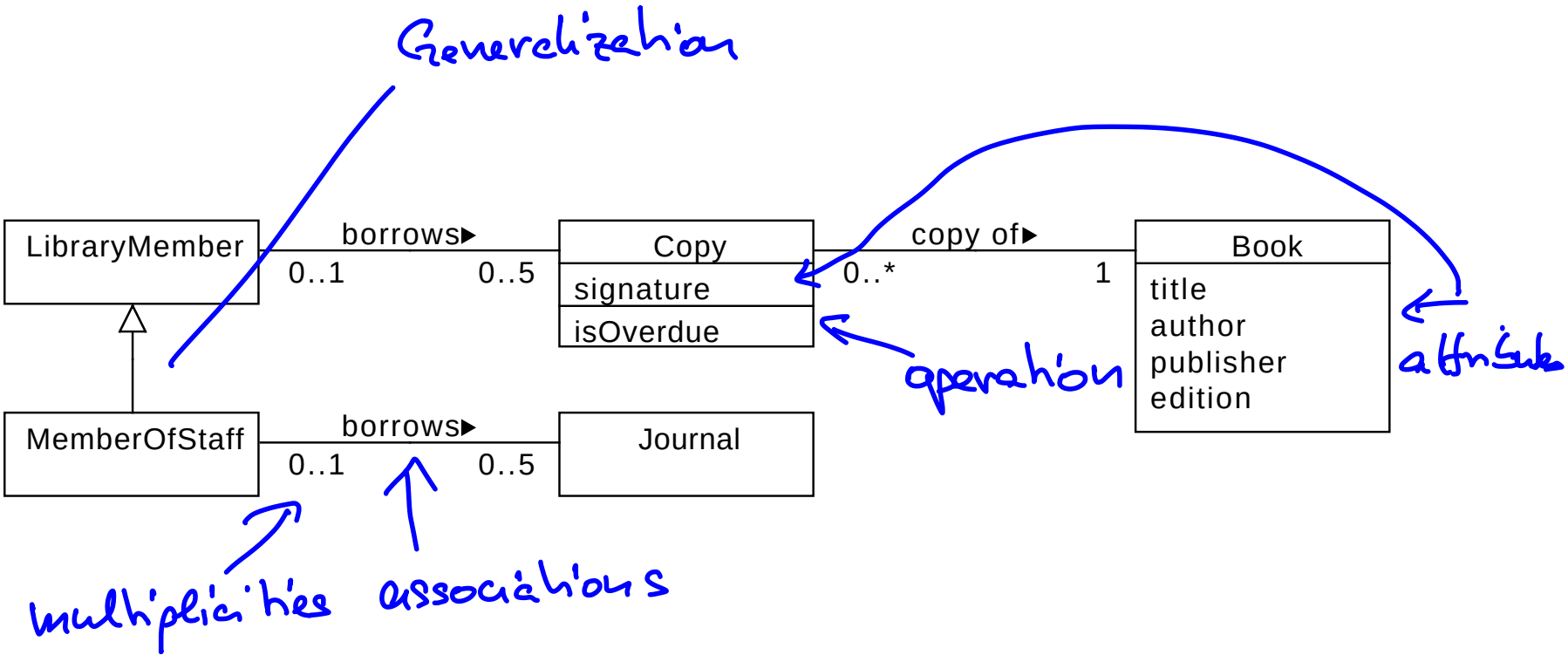
```
public abstract class Component {
 public abstract double cost();
}

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

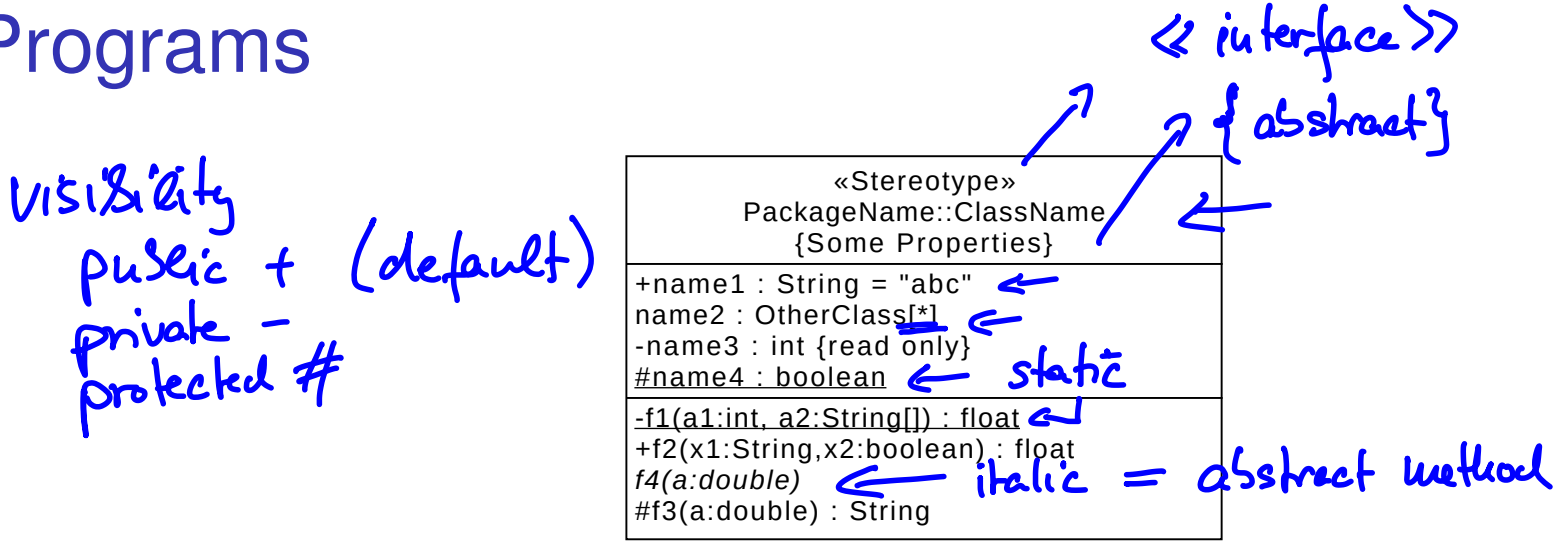
Why a graphical notation?



Class Diagram Example



General correspondence between Classes and Programs




```
package packagename
public class ClassName
{
   private String name1 = "abc";
   public List<OtherClass> name2 = new ArrayList<OtherClass>();
   private int name3;
   protected static boolean navn3;

   private static float f1(int a1, String[] a2) { ... }
   public void f2(String x1, boolean x2) { ... }
   abstract public void f4(a:double);
   protected String f3(double 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; }
}
```

print(person.age)

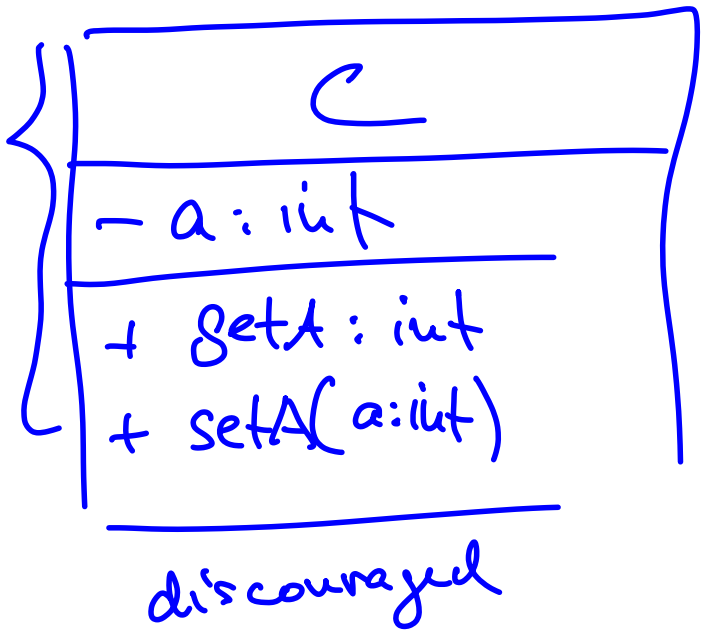
getAge

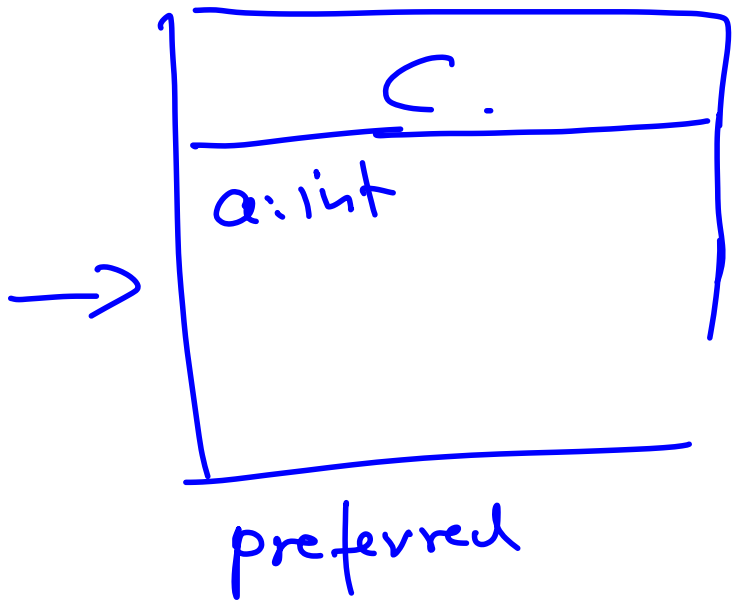
Person

age : int

birthyear : int

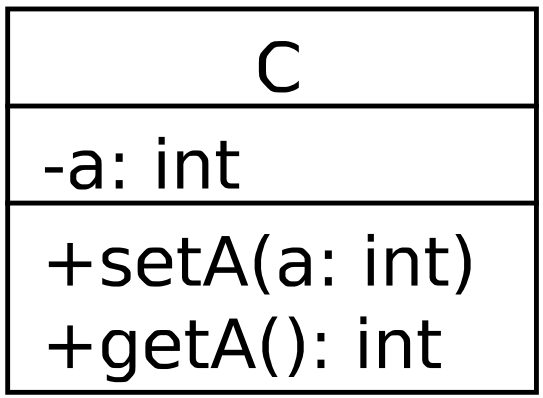
{ age = todays year − birthyear }





Class Diagram and Program Code

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

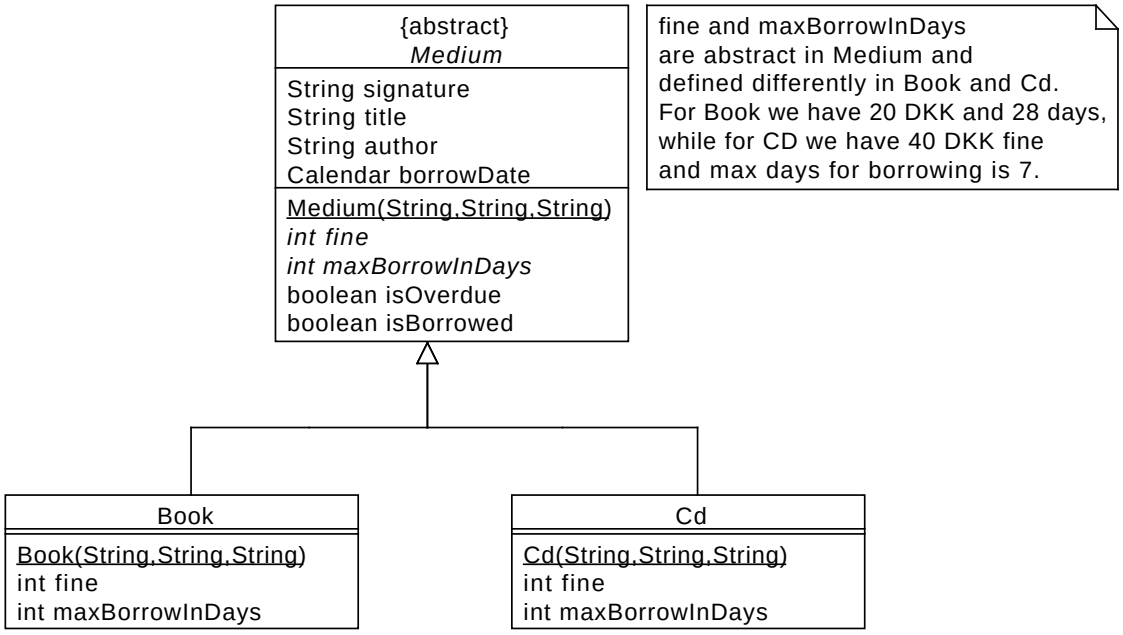


Generalization / Inheritance

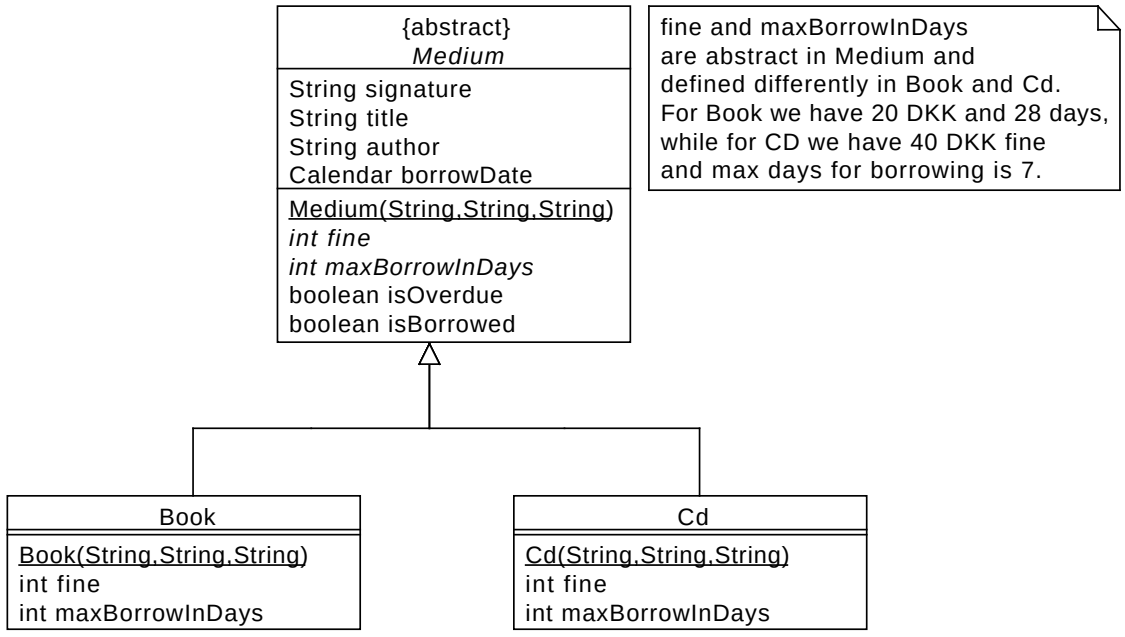
- Programming languages like Java: Inheritance

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

- UML: Generalization / Specialization



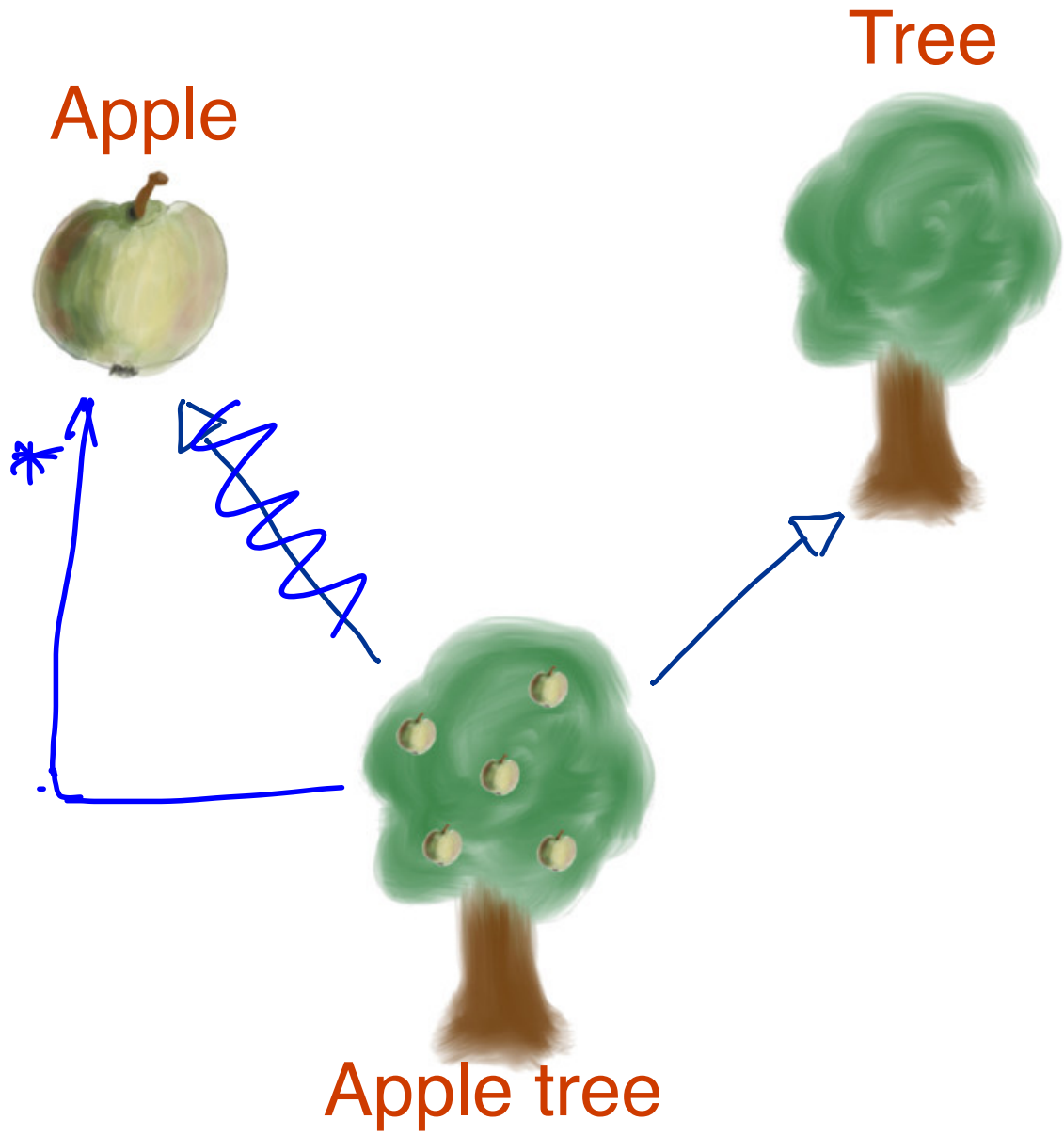
Generalisation Example



Liskov-Wing Substitution Principle

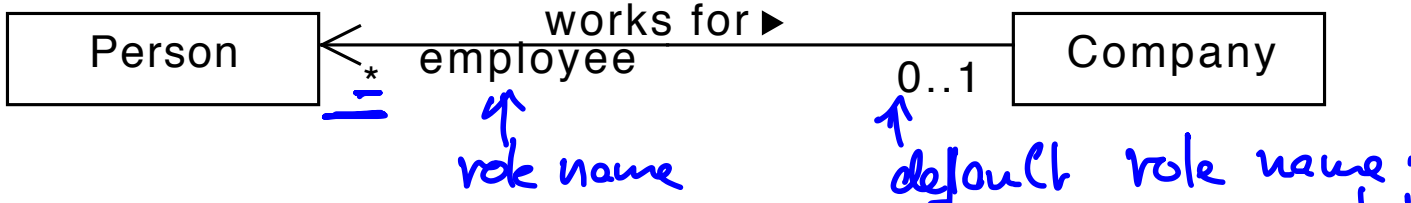
> *"If S is a subtype of T, then objects of type T in a program may be replaced with objects of type S without altering any of the desirable properties of that program (e.g., correctness)."*

Appletree



Associations between classes

- Unidirectional (association can be navigated in one direction)



  - Company has a field employees

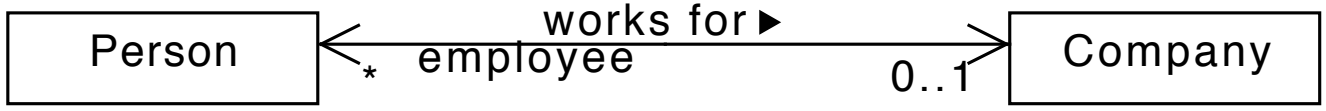
```
public class Person
{
    ....
}
```

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

List

Associations between classes

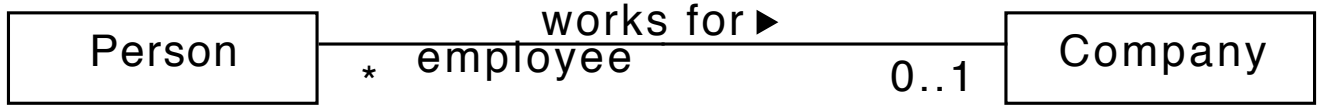
- Bidirectional (association can be navigated in both directions)



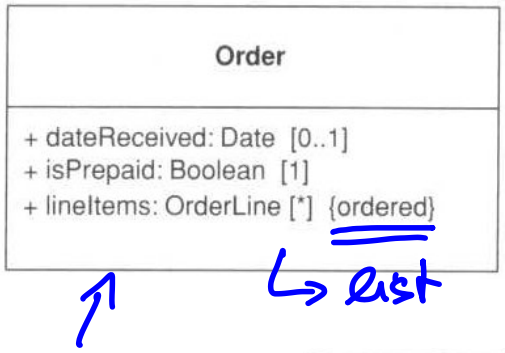

```
public class Person
{    ....
     private Company company;
     public getCompany() {
       return company;
     }
     public setCompany(Company c) {
       company = c;
     }
     ....
}
```

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

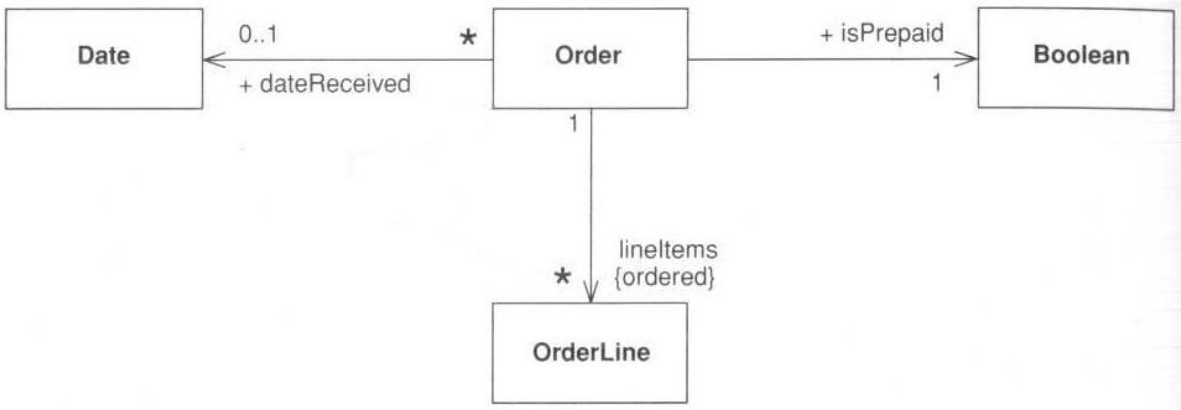
- Bidirectional or no explicit navigability
  - no explicit navigability ≡ no fields



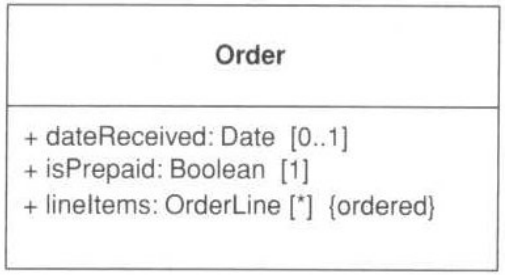
Attributes and Associations



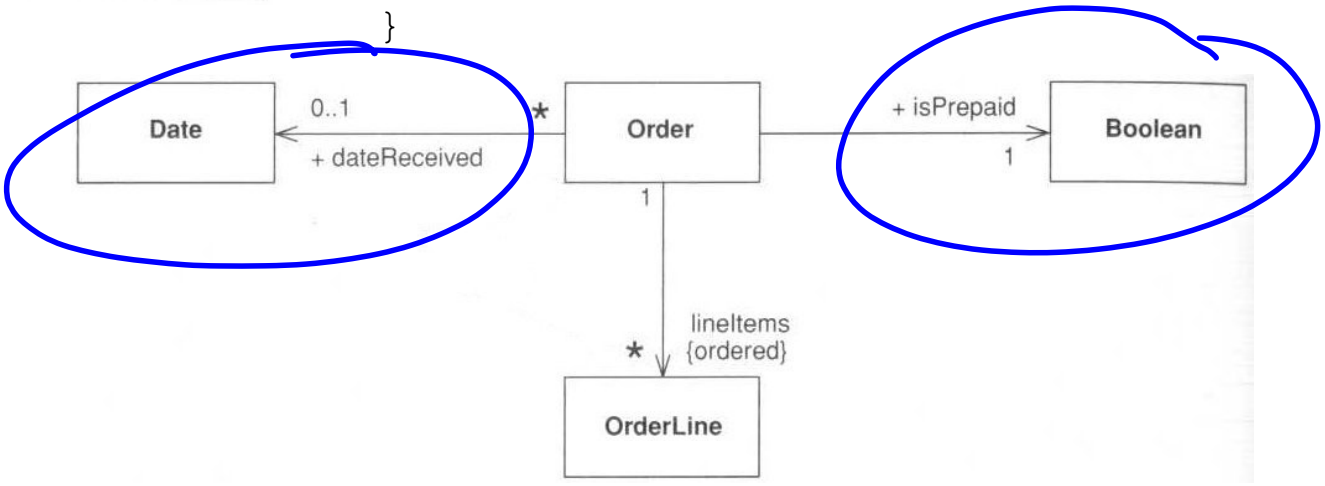

```
public class Order {
  private Date date;
  private boolean isPrepaid = false;
  private List<OrderLine> lineItems =
    new ArrayList<OrderLine)();
  ...
}
```

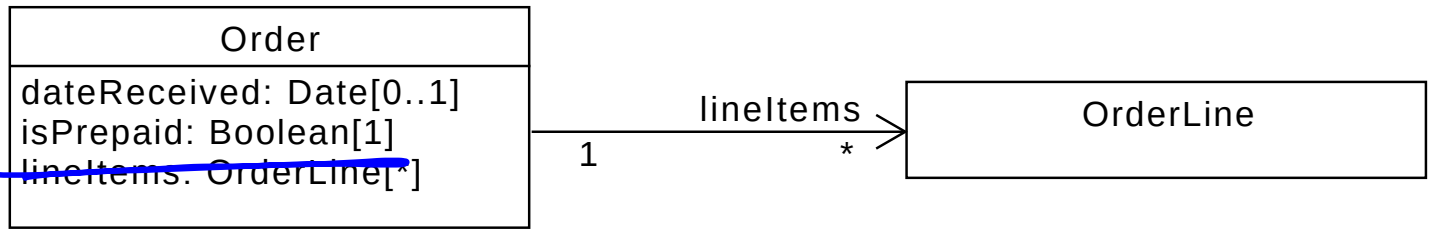


Attributes and Associations




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

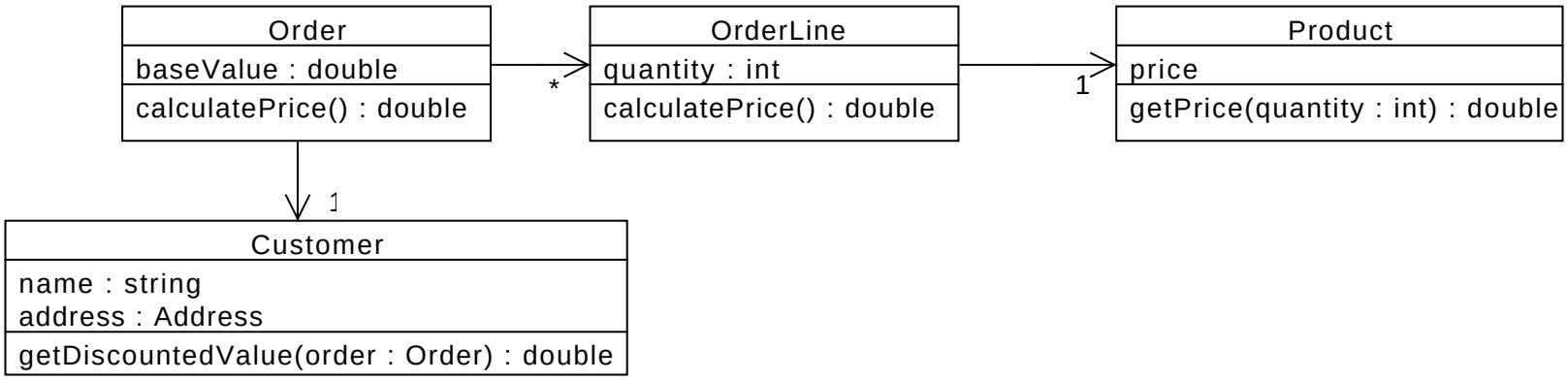




Contents

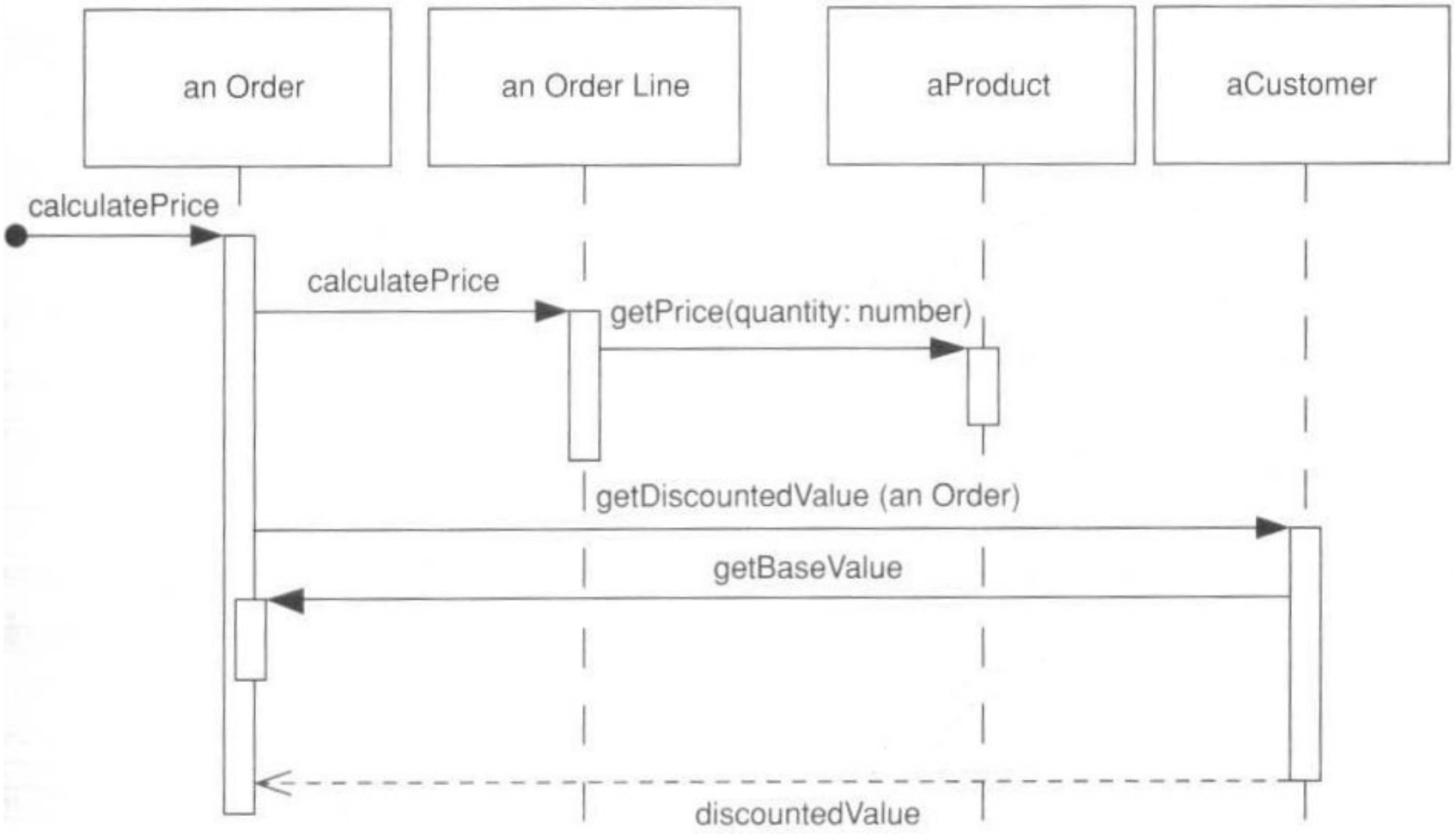
Sequence Diagram: Computing the price of an order

- Class diagram

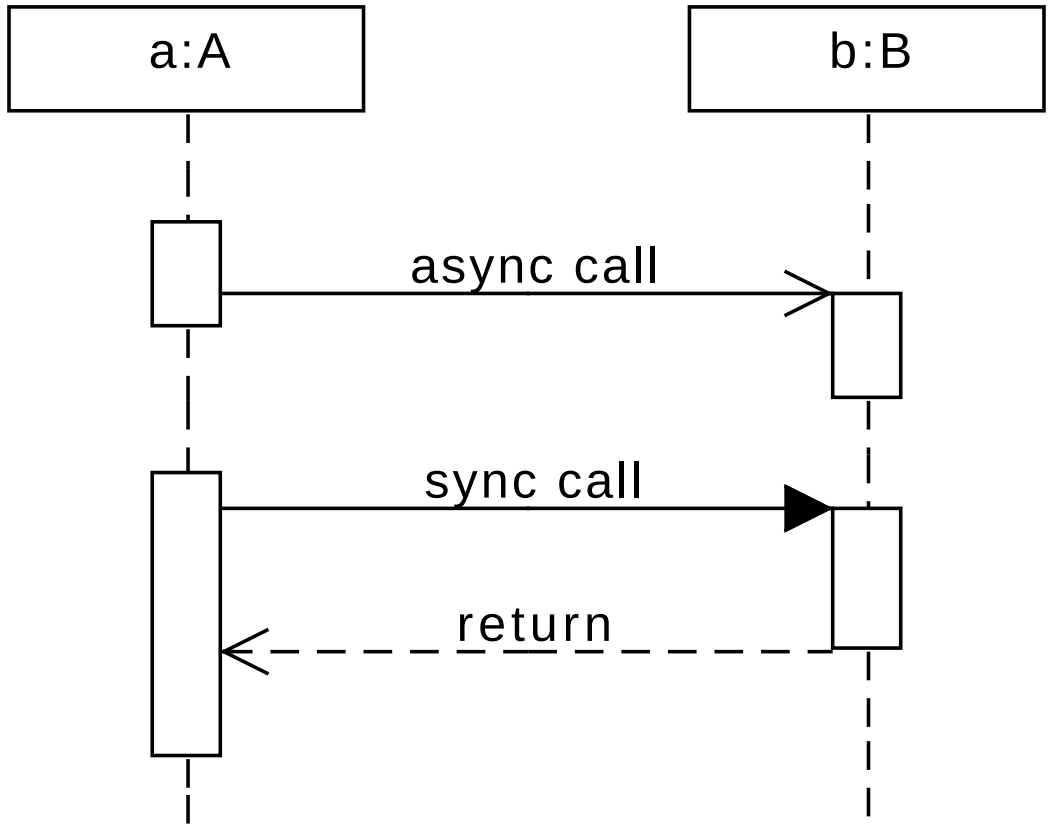


- Problem:
  - What are the operations doing?

Sequence diagram



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 6 (this week) – 8:
  - 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 8—13:
  - Implementation
  - Systematic tests and design by contract
- 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?
  - Design and implement a project planning and time recording system
  - UI required, but not a graphical UI; database / persistency layer is not required
- Deliver
  - Week 8: report describing the requirement specification and design (**mandatory**; contributes to the final grade)
  - 18.4.: First draft of the impementation and tests (**not mandatory**; won't be graded but you will get feedback)
  - Week 13:
    - report on the implementation and tests (**mandatory**; contributes to the final grade)
    - Standalone **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) (**mandatory** contributes to the final grade)
    - demonstration in front of TA's (**participation mandatory**; does not contribute to final grade)
- More detail on CampusNet

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. needs to 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 copying of text without naming the sources is viewed as cheating*
- In case of questions with the project description send email to `huba@dtu.dk`

Week 6–8: Requirements and Design

Recommended (but not mandatory) Design process

1. Create glossary, use cases, and domain model
2. Create user stories based on use case scenarios
3. Create a set of initial classes based on the domain model
   $\rightarrow$ 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
   c) Document the scenario using a sequence diagram (only if needed to document the execution)
3. Repeat step 2 with the other user stories

Apply the Pareto principle: 20% of the work gives 80%: Include the important details but don't try to make your model perfect.

Learning objectives of Week 6—8

- 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)
- I don't expect you to create perfect models
  - It is perfectly okay if your final implementation does not match your model
  - By comparing your model with your final implementation, you learn about the relationship between modelling and programming

Week 8—13

Recommended (but not mandatory) 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) Implement the user story test-driven, creating additional tests as necessary, **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
      - → Criteria: ideally 100% code coverage of the business logic (i.e. application layer) 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 models, report, and the implementation
- Evaluation criteria
  - In general: correct use and understanding of the techniques introduced in the course
  - Implementation: good architecture (e.g. use of layered architecture), understandable code and easy to read (e.g. short methods, self documenting method names and variable names, use of abstraction)
  - Rather focus on a subset of the functionality with good code quality than on having everything implemented but with bad code quality