Next Thursday

- “Extended Modelling Notations, Experiment”

- Expressing scenarios that must/must not occur
- Analysing models (E.g. verification)
- Non-UML notations

What about the Experiment?

- Second part of the lecture: Experiment
- Connected to my (Grischa) research:
  Are some notations harder/easier to understand than others?
- Participation is **voluntary**…but would really help me!

- And: Similar question style as voluntary exam III. So, it's a good practice!

Object-oriented System Development

Lecture 7

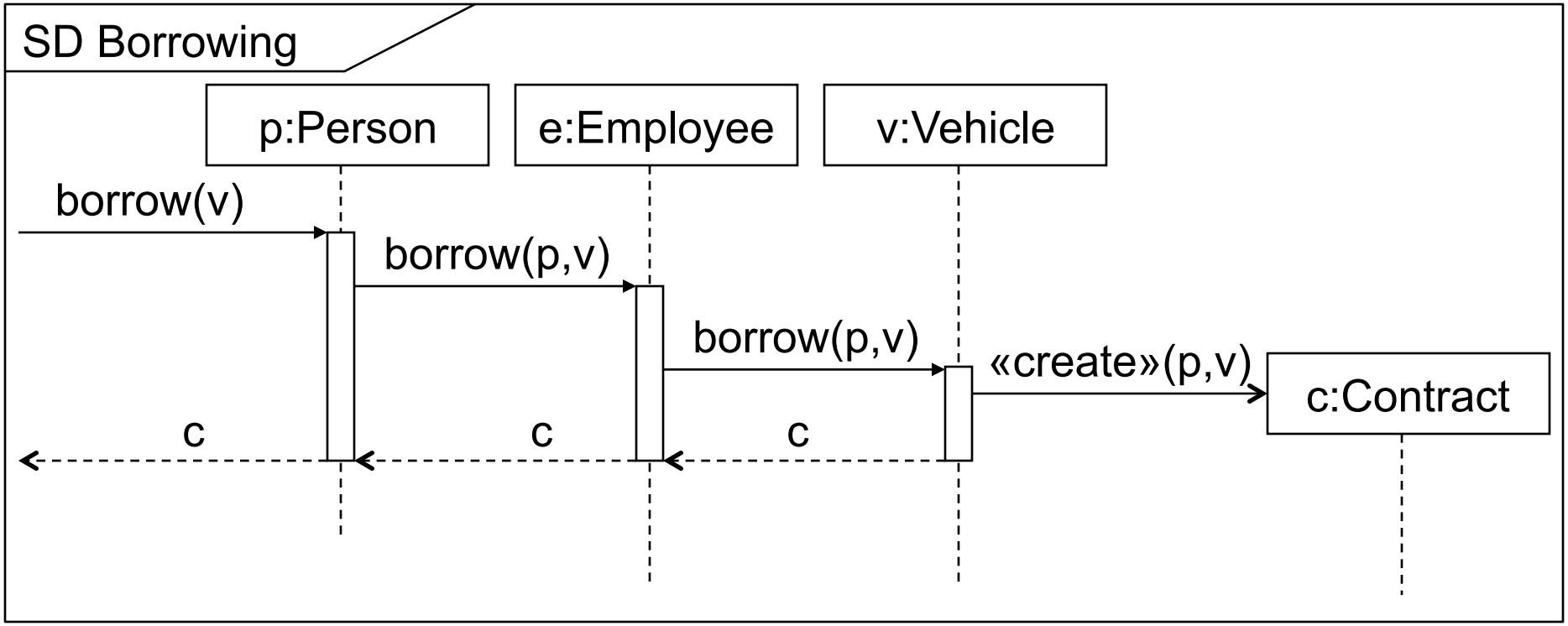
Sequence Diagrams

Learning Outcomes

- …be able to create UML Sequence Diagrams (Lifelines, different message types, different combined fragments, conditions)
- …be able to reflect on the complexity of Sequence Diagram. When is it suitable to have a higher/lower abstraction/level of detail?
- …be able to create Sequence Diagrams given component and interface definitions and/or Use Cases
- …be able to describe a process to systematically refine your system design starting from sequence diagrams with abstract components down to actual façade classes, interfaces, and classes within your component
- …be able to argument why you use/don't use certain parameter types in your component interfaces

Sequence Diagrams

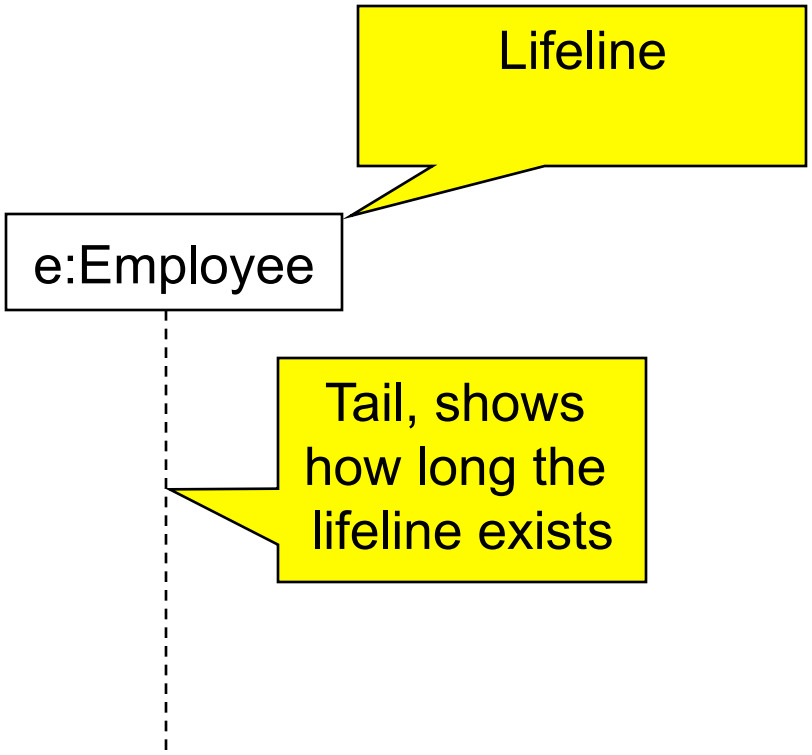
- Concentrate on control flow



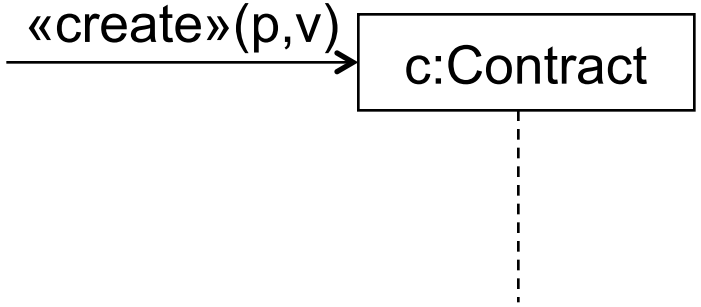
Sequence Diagrams

- Concentrate on control flow, show invocations of methods & operations
- Diagrams do *not* show
  - Computations that take place
  - Most of the data flow
- Responsibilities of classes in context
  - More readable than a textual description
- Time runs from top to bottom in diagram!

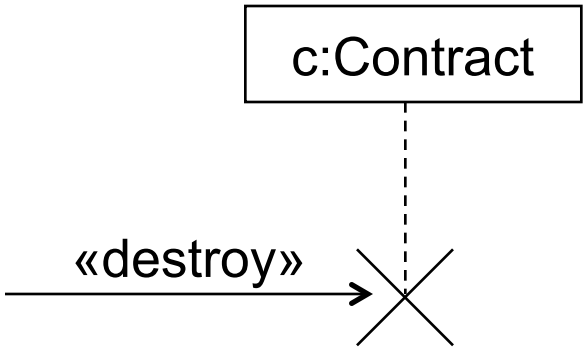
Basic Sequence Diagrams: Objects



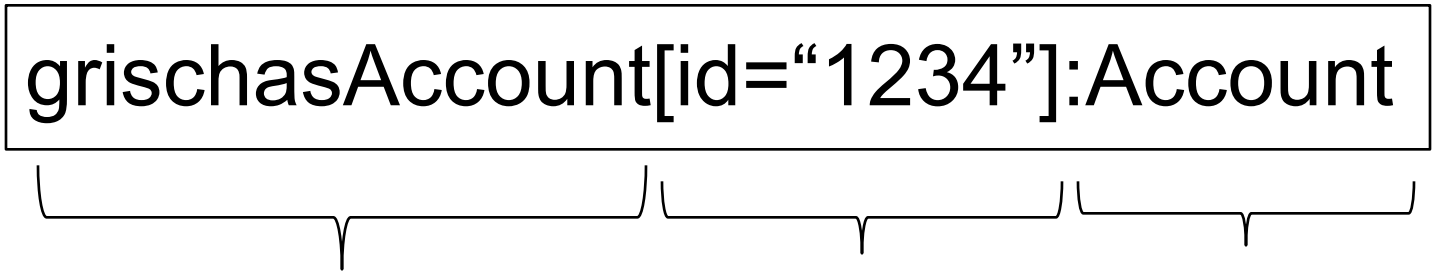
Objects can be created:



… and destroyed:



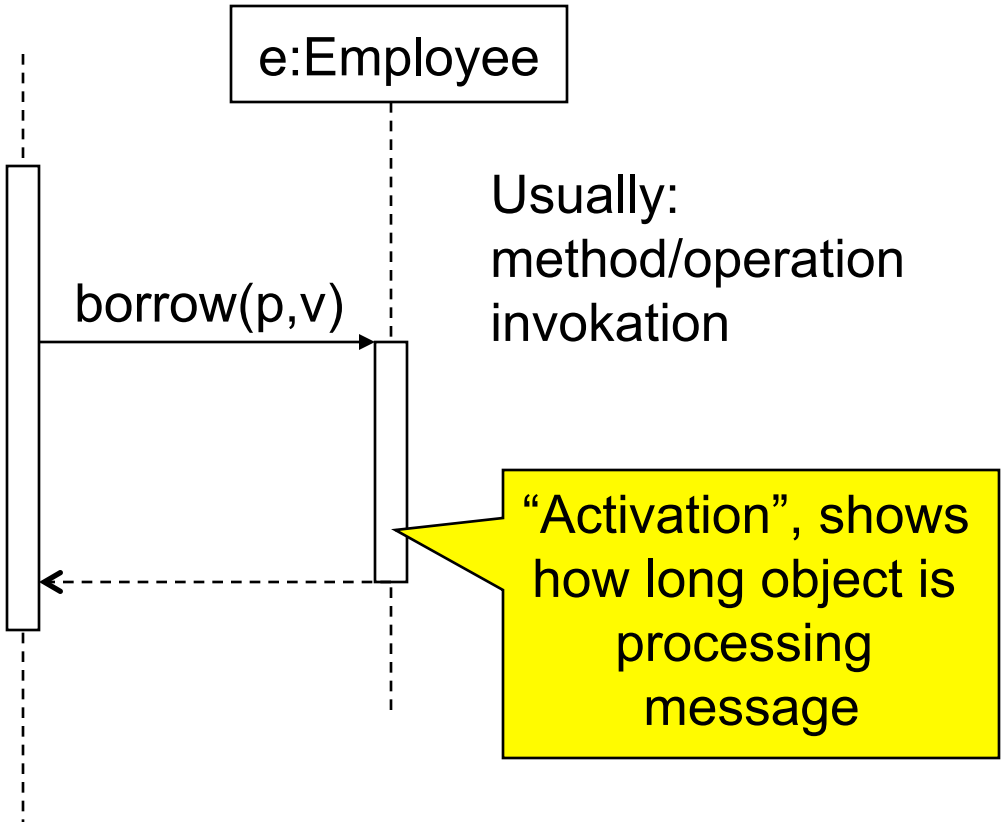
Lifelines



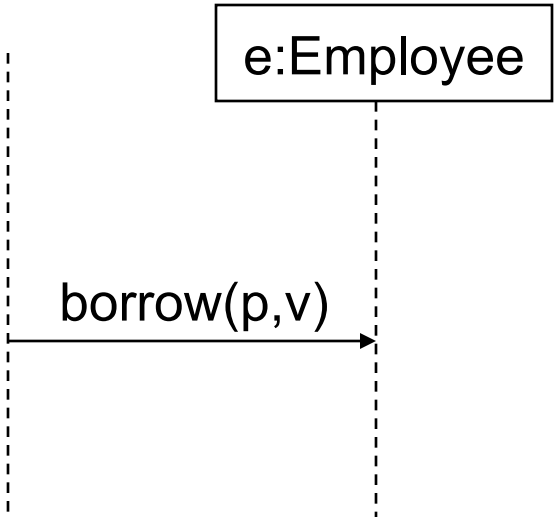
name of the classifier of which the lifeline represents an instance

refer to the lifeline within the interaction

Basic Sequence Diagrams: Messages

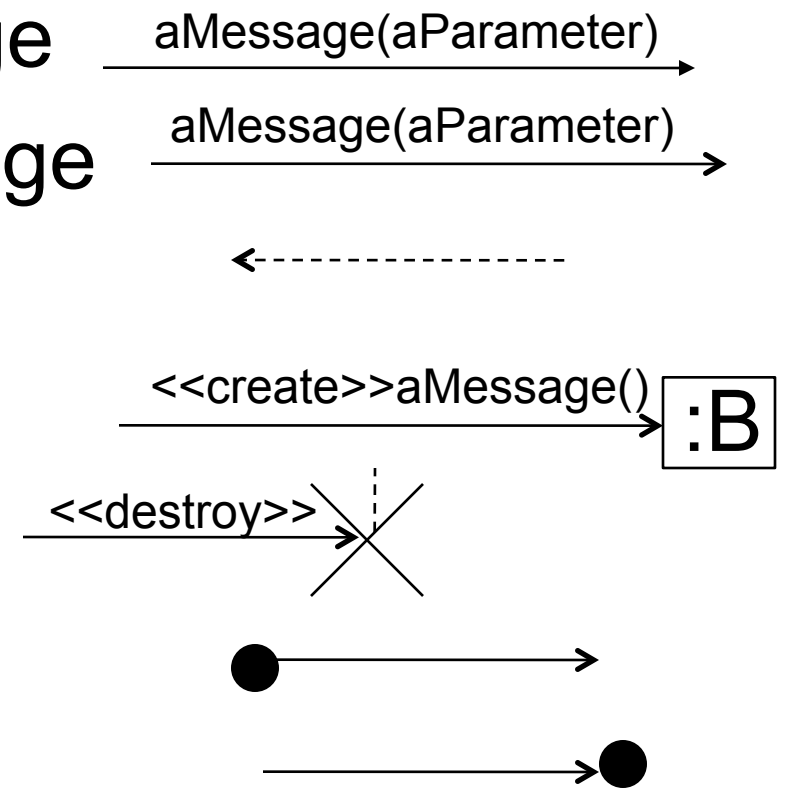


Activations are often left out in diagrams:



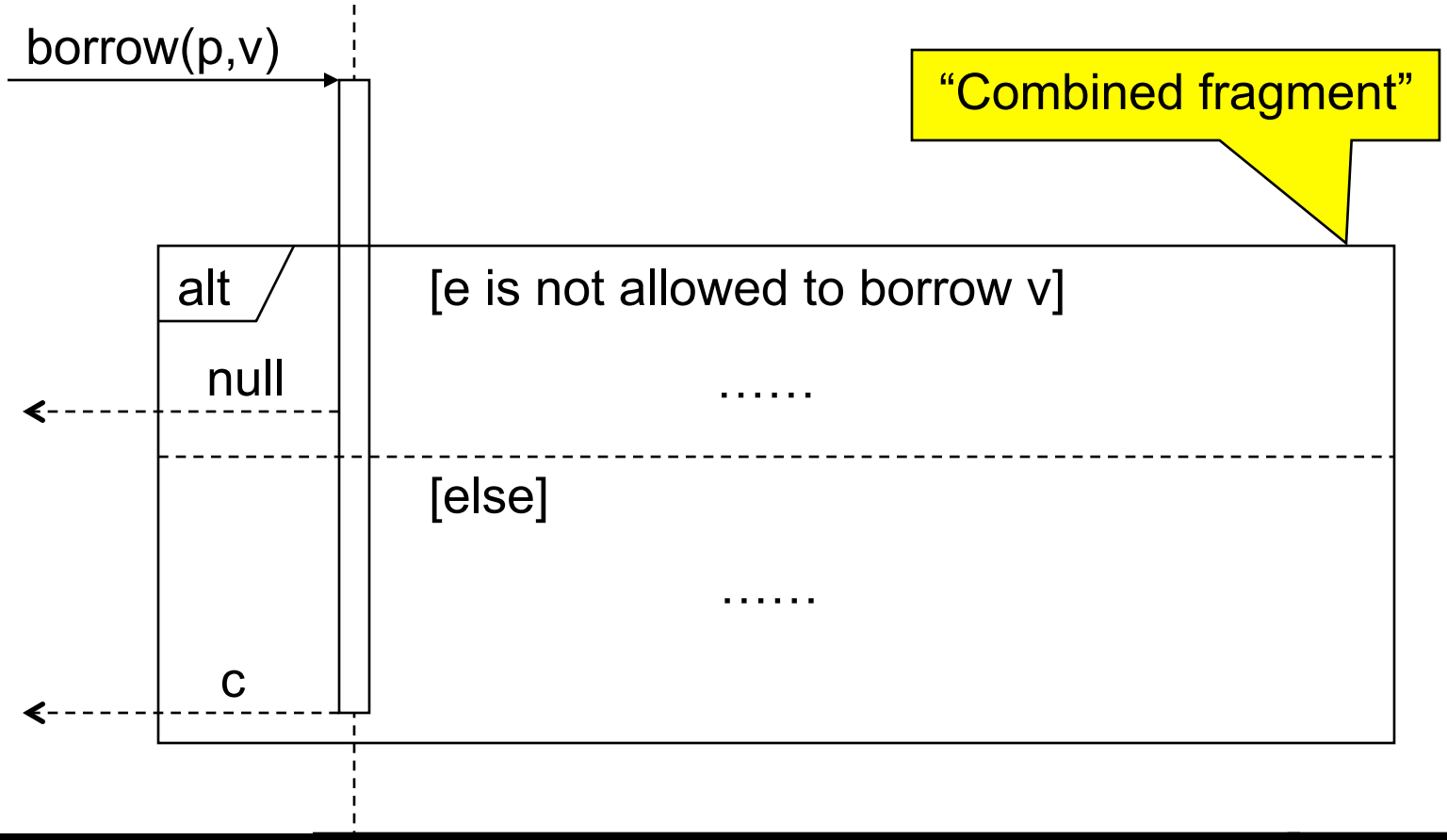
Messages

- Synchronous message
- Asynchronous message
- Message return
- Object creation
- Object destruction
- Found message
- Lost message

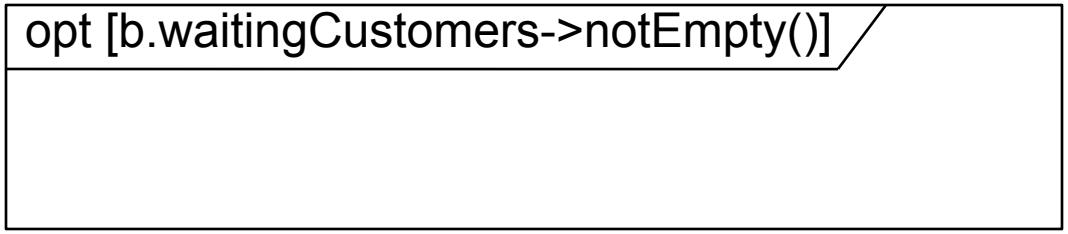


Basic Sequence Diagrams: Alternatives

- Choose between two (or more) possible scenarios:

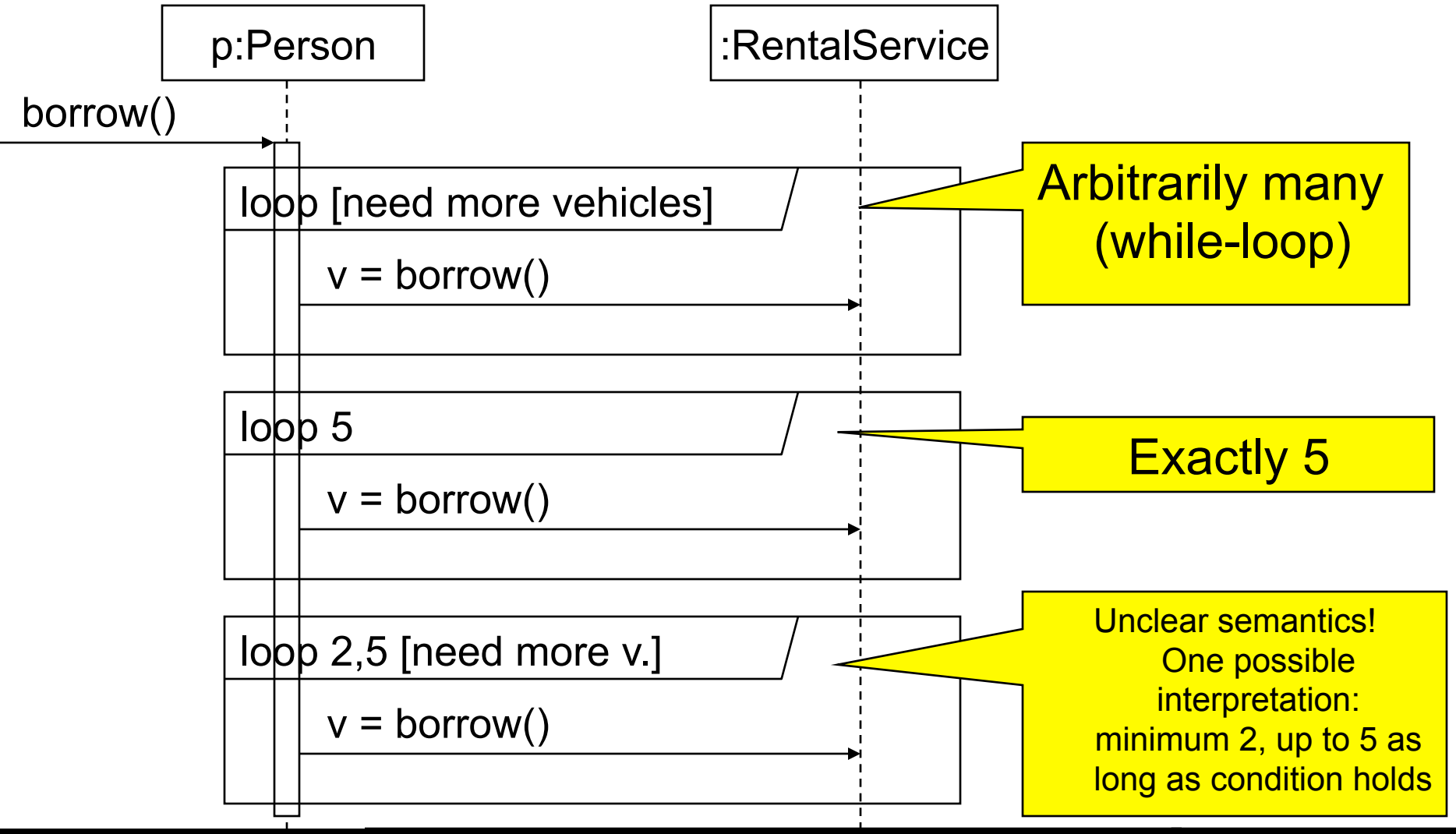


Other Combined Fragments



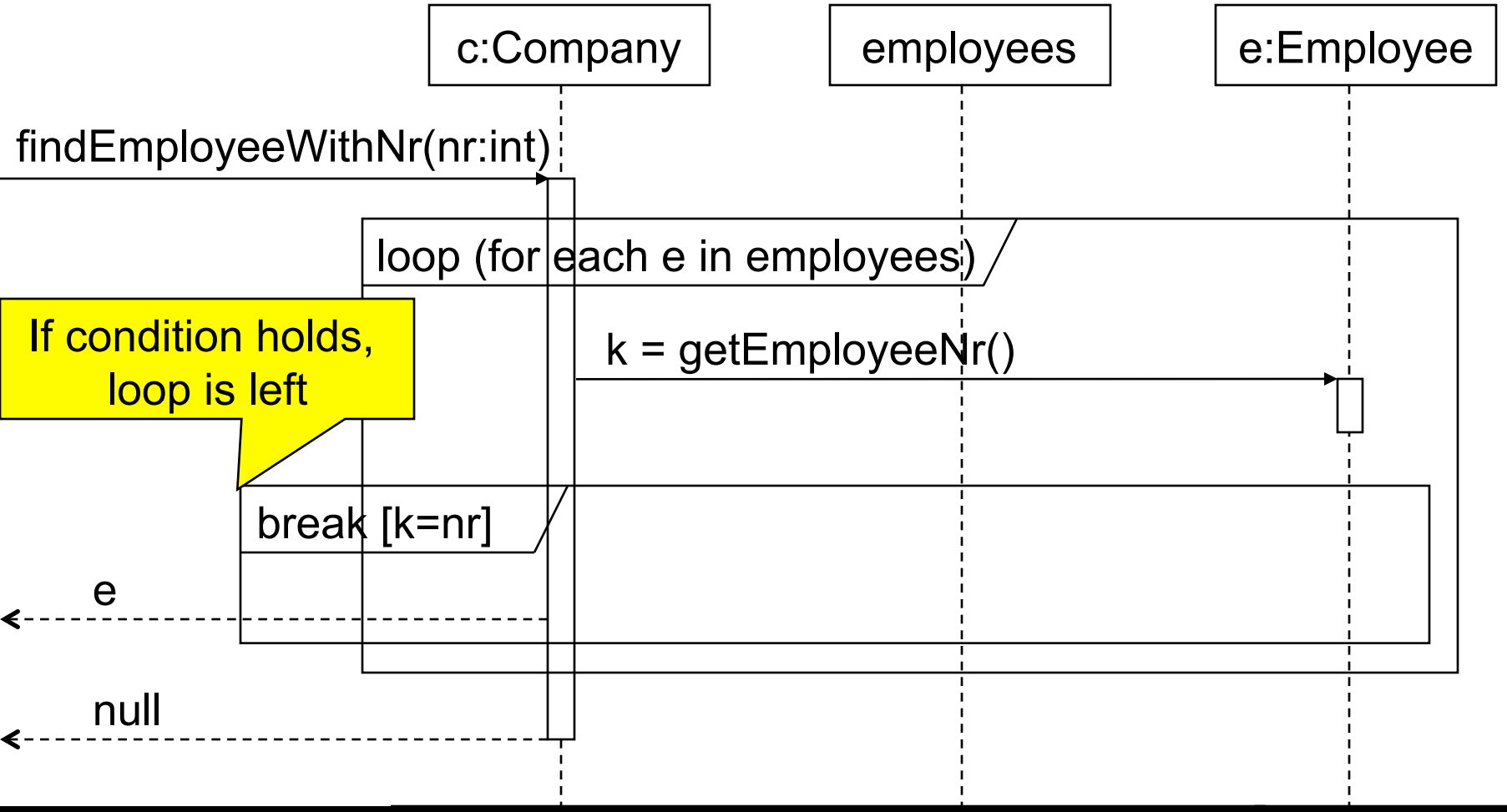
- neg
- assert
- …
- See the UML spec!

Loops


p:Person
:RentalService
borrow()
loop [need more vehicles]
v = borrow()
Arbitrarily many (while-loop)
loop 5
v = borrow()
Exactly 5
loop 2,5 [need more v.]
v = borrow()
Unclear semantics! One possible interpretation: minimum 2, up to 5 as long as condition holds

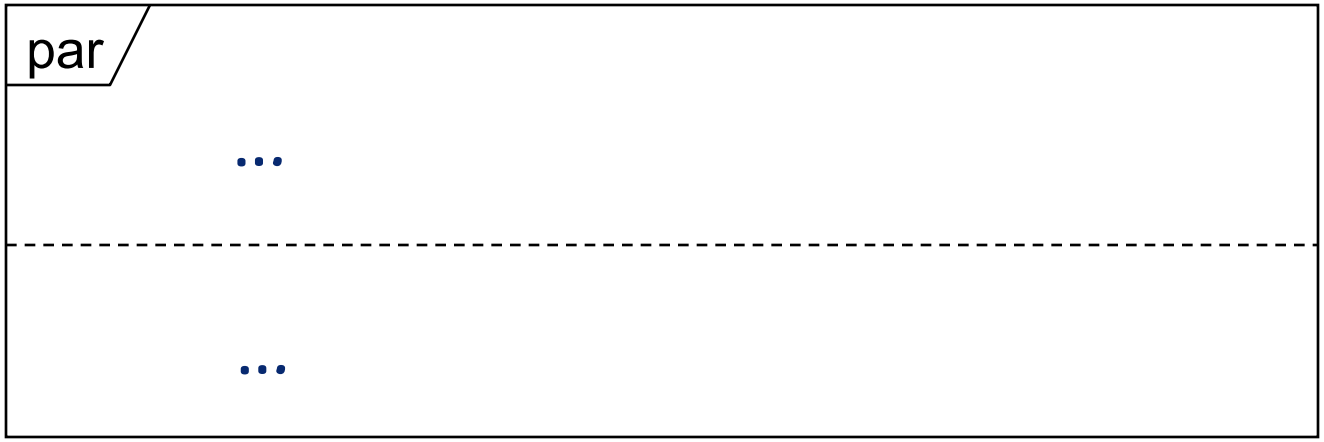
Combined Fragments: Break

- Variant of “opt”: Leave enclosing blocks if condition holds



Combined Fragment: Parallelism

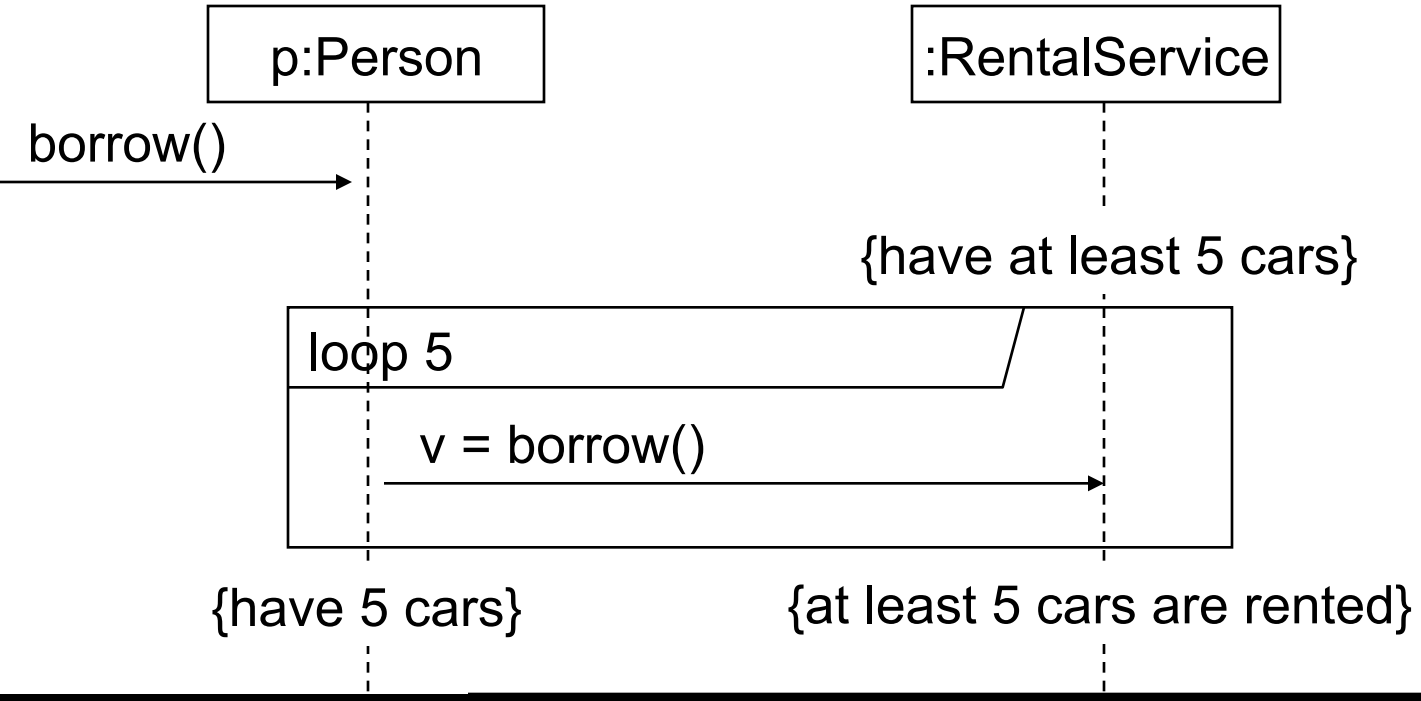
- Do 2 or more things in parallel (order is unspecified)
- Similar to activity diagrams



- Further keywords: seq, strict, critical

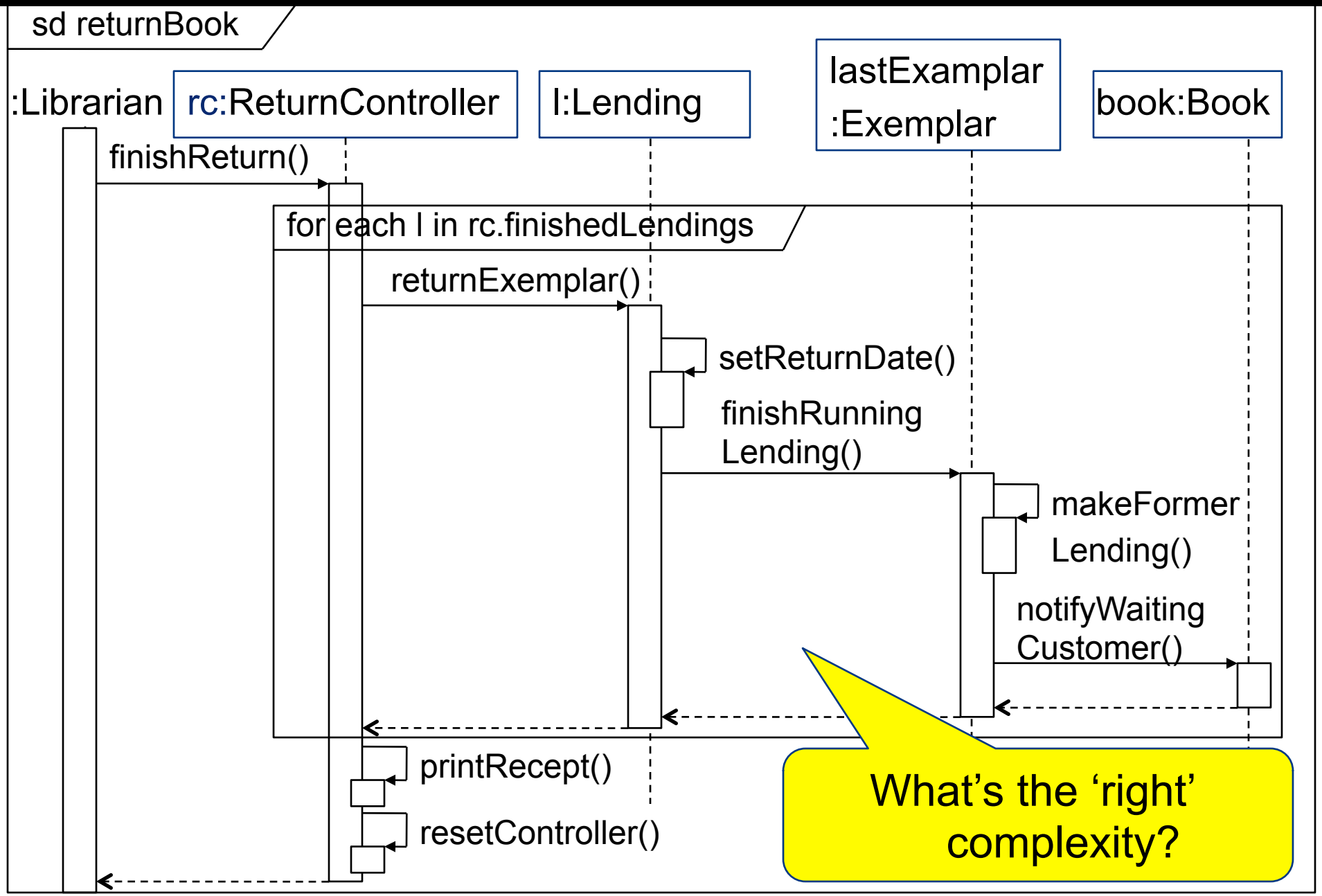
State Invariants

- Express that some property is supposed to hold at a point
- Documentation + consistency checks



What are Interaction Diagrams good for?

- Distributing responsibilities, designing interaction between systems or objects
- Documentation
- Testing, comprehension: Diagrams for particular scenarios (traces) can be created automatically from code


sd returnBook
:Librarian
rc:ReturnController
l:Lending
lastExamplar
:Exemplar
book:Book
finishReturn()
for each l in rc.finishedLendings
returnExemplar()
setReturnDate()
finishRunning
Lending()
makeFormer
Lending()
notifyWaiting
Customer()
printRecept()
resetController()
What's the 'right' complexity?

Complexity of Interaction Diagrams

- Big interaction diagrams are very hard to read
- Important to choose right level:
  - Abstraction: how many details?
  - Generality: how many cases?
- Decide which aspects a diagram is supposed to show!
  - Hide everything else

How many Details?

- Unfold method calls?
- UML “ignore” feature (Unclear semantics!)
- Sub-diagrams? (“ref” Combined Fragment)

- Also: Is it important to be accurate?
- Who do you target with the diagram?
  (Manager, Developer, Customer?)

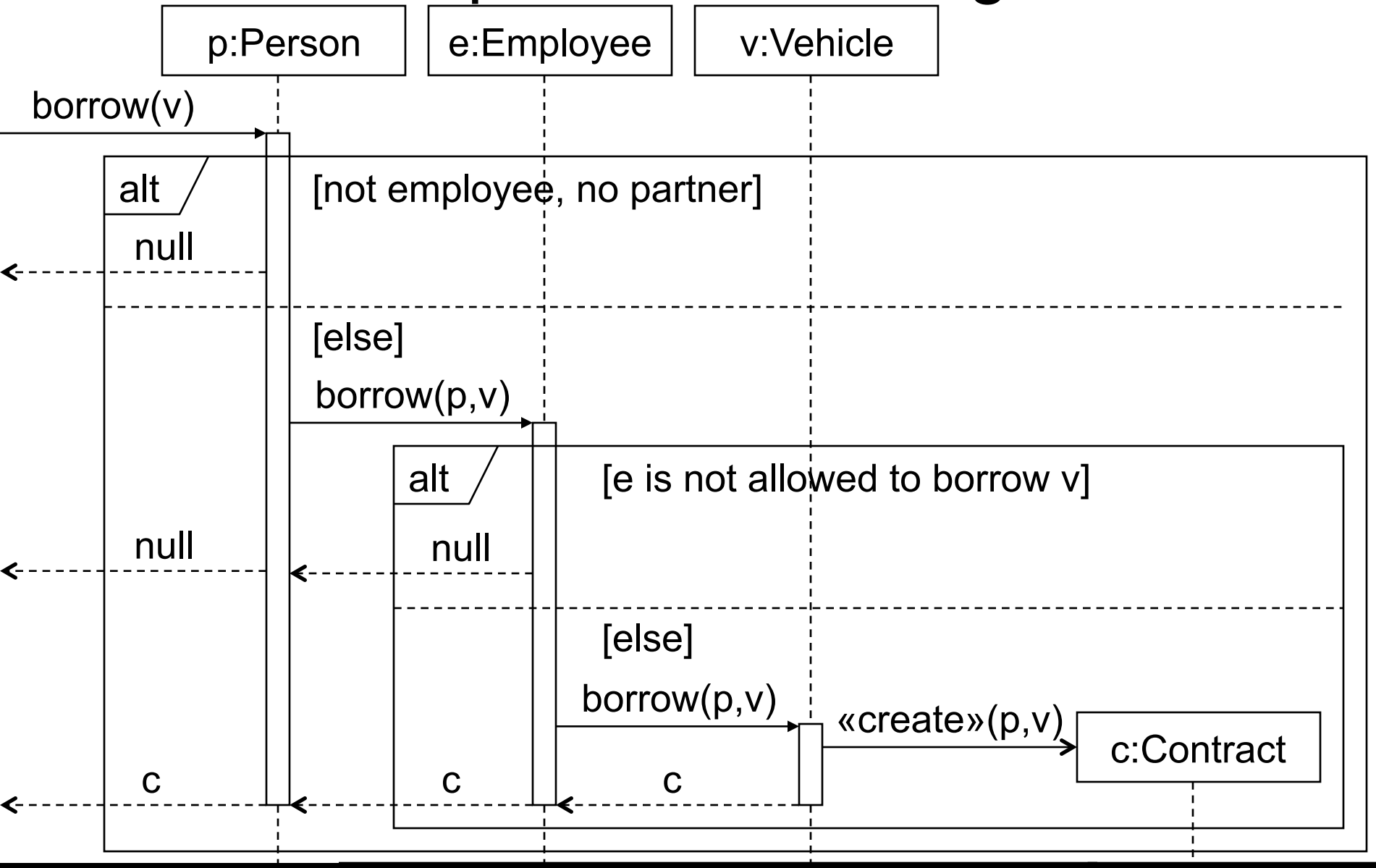
How many Scenarios?

- One general diagram

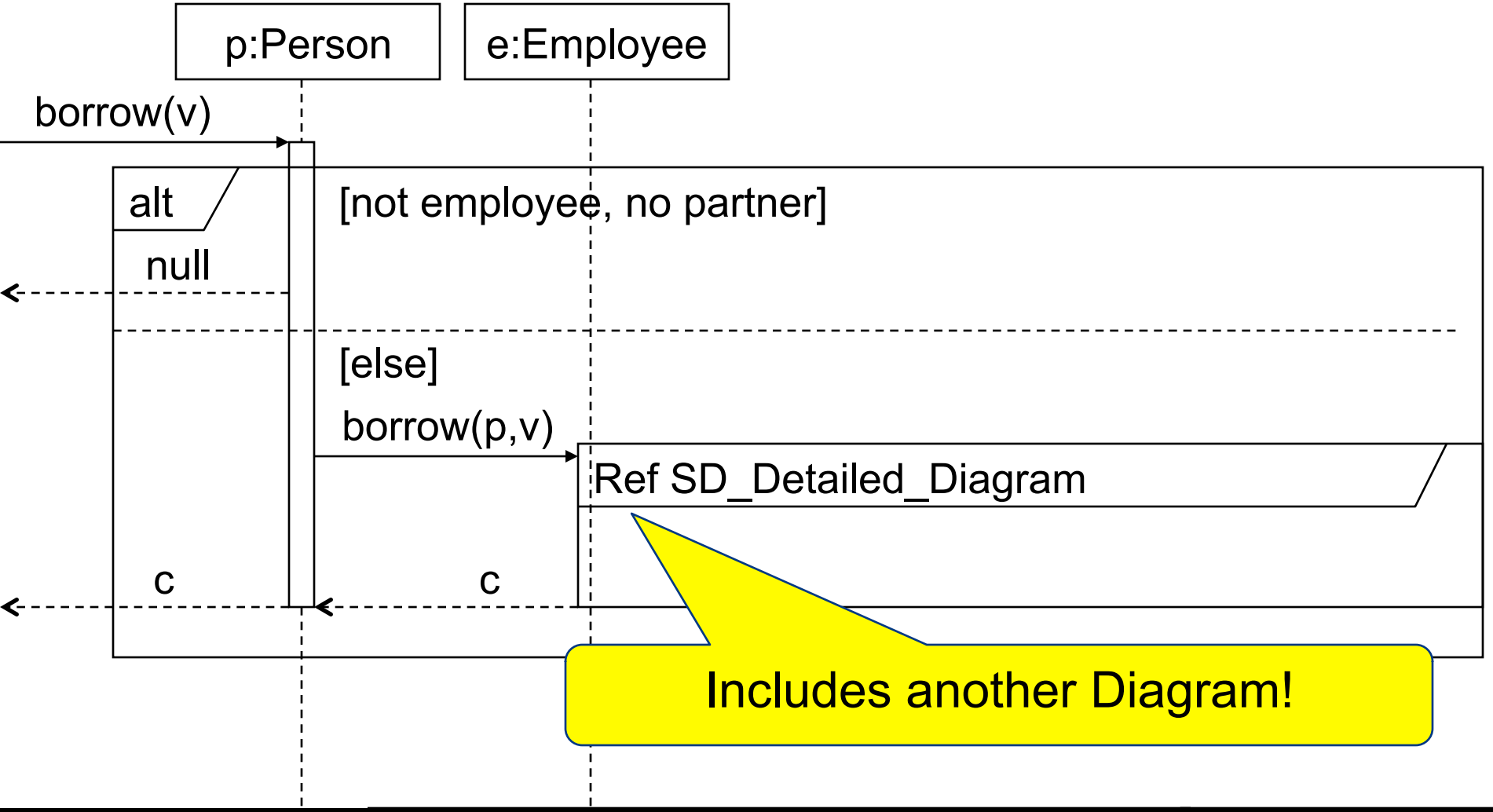
- Many specialised diagrams

- Similar to use cases (vertical splits)
- Handling too many cases (e.g. errors) clutters diagrams

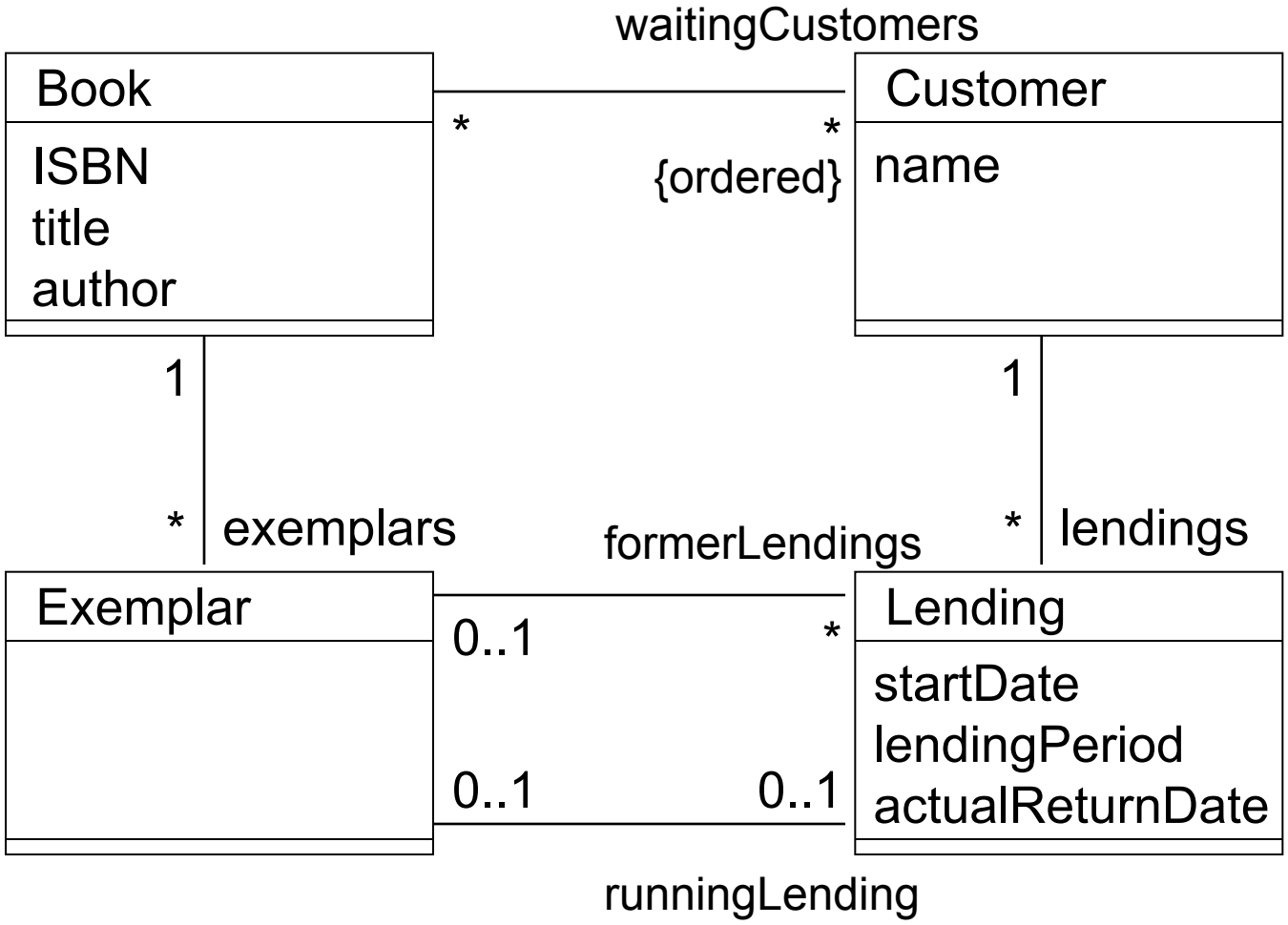
Example: General Diagram



Referencing other Diagrams



Problem Domain: A Library (once more)



Use Case: Returning Books

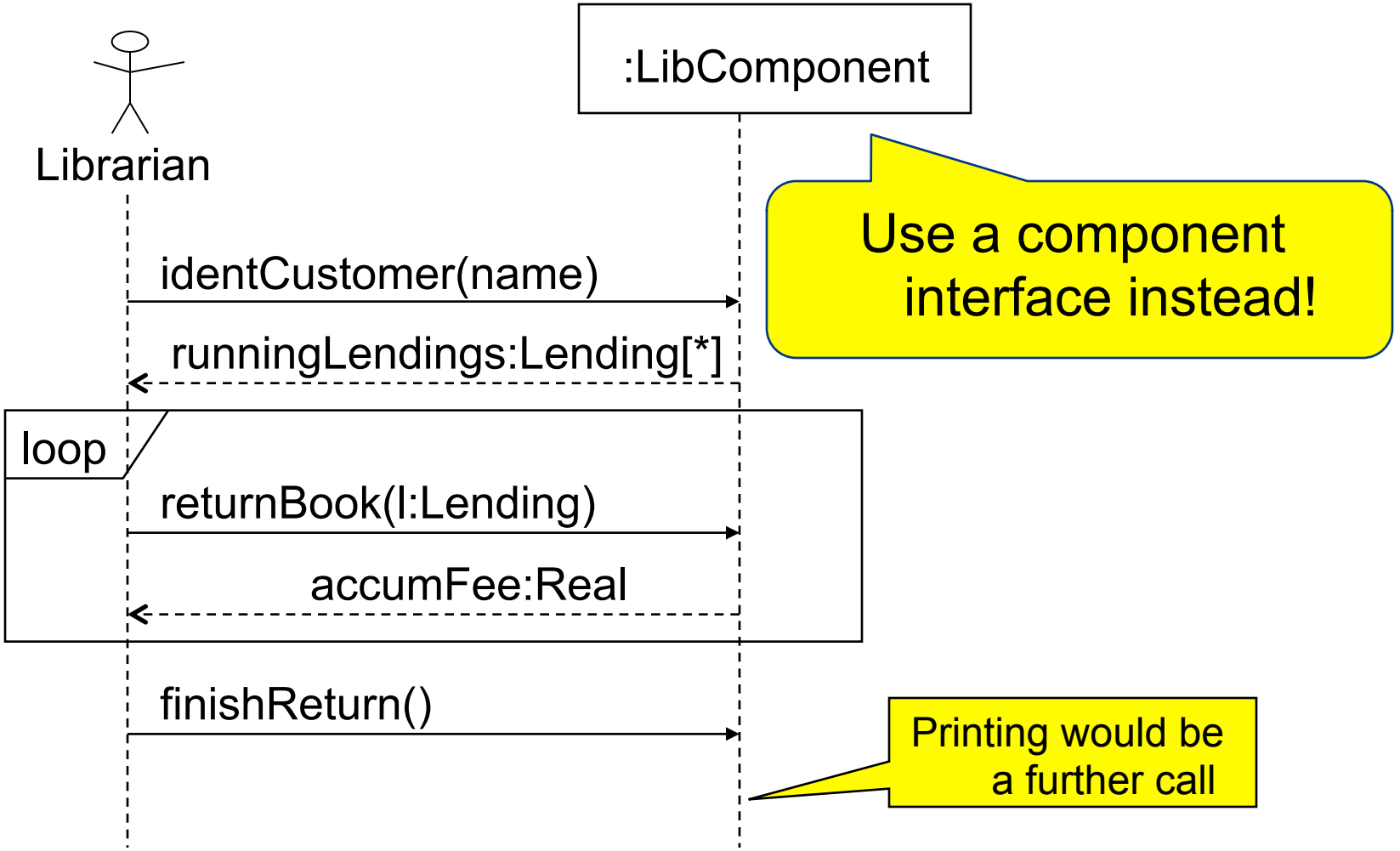
- Actor: Librarian
- Goal:
  Register books returned by customer
- Description:
  The librarian enters the name of a customer and selects one or more books as returned. The system registers the books as returned, prints a receipt with the fee that the customer has to pay for late books, and notifies customers that are possibly waiting for the returned books.

Write a Detailed Use Case and a System Sequence Diagram!

Use Case: Returning books

1. Librarian starts return of books
2. System prompts customer name
3. Librarian enters customer name
4. System validates name, retrieves lent books
5. System displays list of lent books
6. WHILE (further books to be returned)
    1. Librarian selects a returned book
    2. System checks the lending period
    3. System displays accumulated fee for late books
7. Librarian finishes the book return
8. System marks the selected books as returned
9. System prints a receipt with the fee for late books
10. System prints notifications for waiting customers

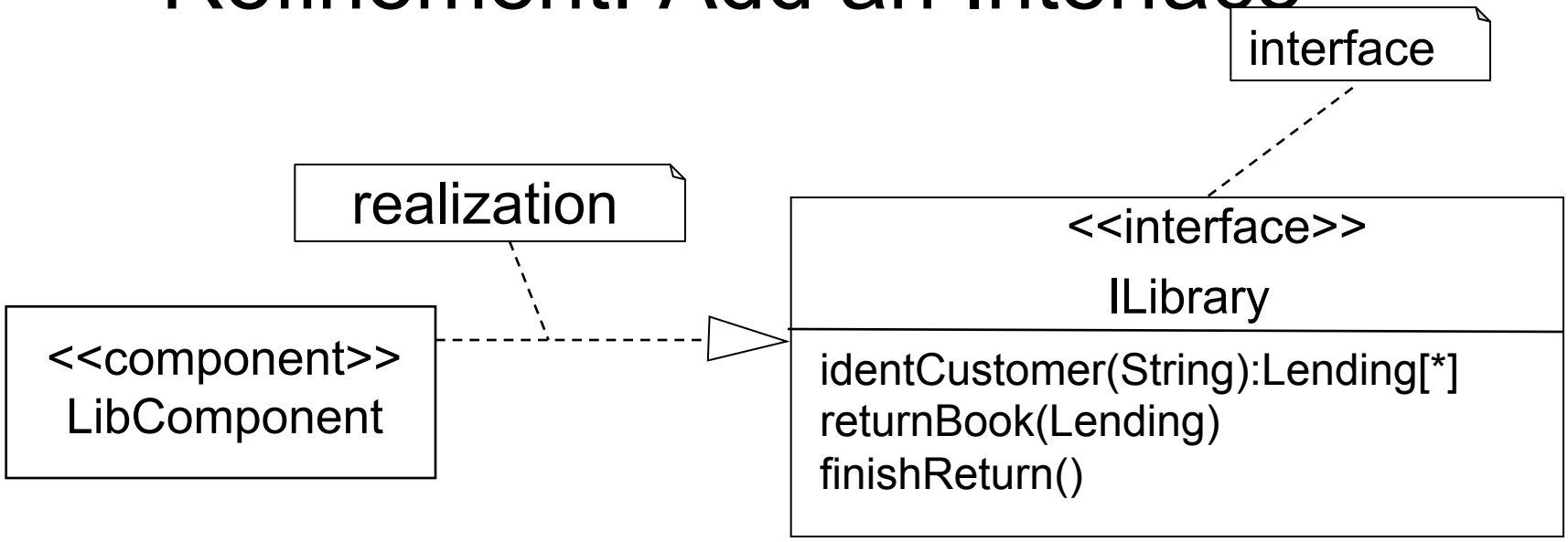
System Sequence Diagram



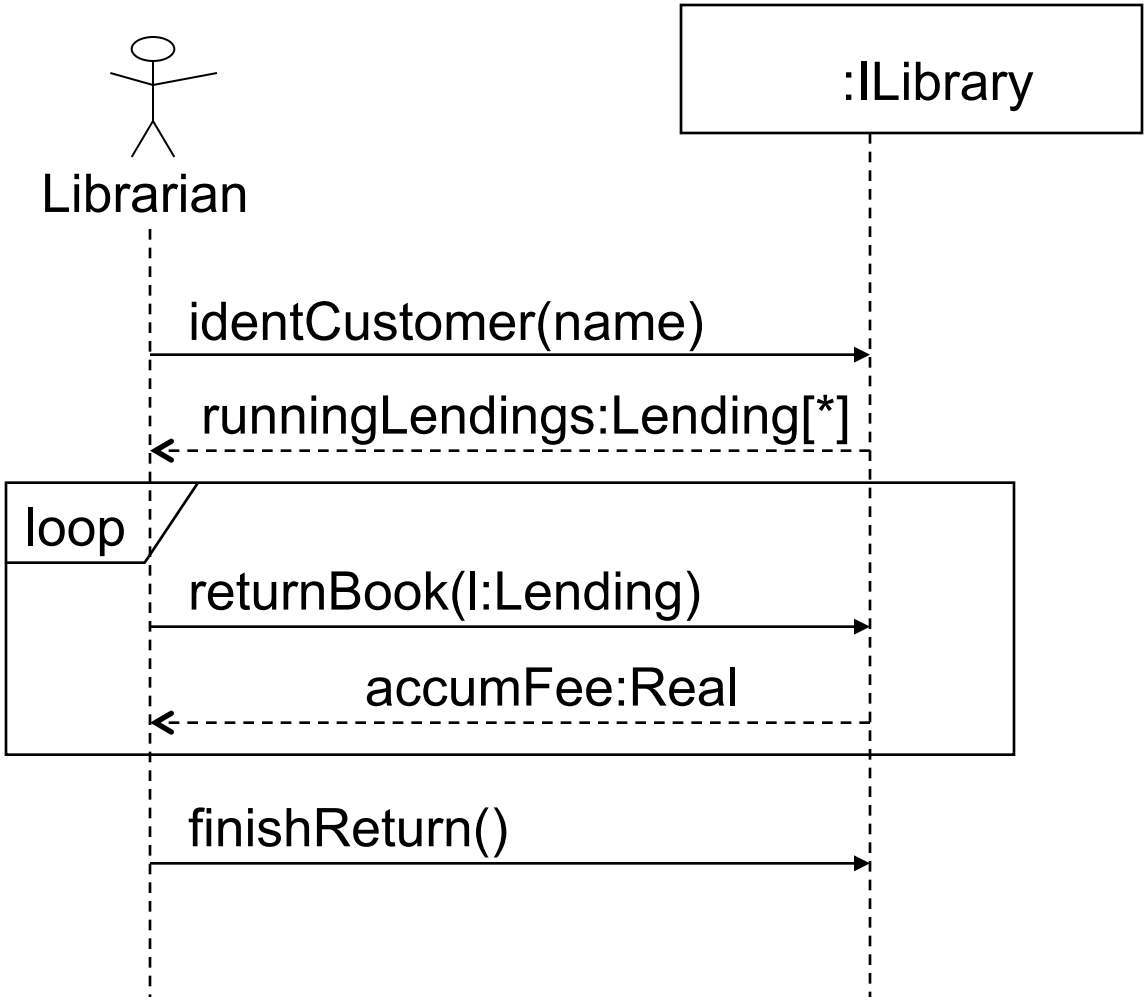
Refinement of (System) Sequence Diagrams

- One possibility:
  1. Draw a sequence diagram that depicts communication between user and components (or between components)
  2. Divide each component into multiple interfaces → Refine sequence diagrams to communicate with the interfaces instead
  3. Within the component, start with your domain model and add façade classes for each interface
  4. Later on: possibly refactor the class diagram to include further classes that implement the actual functionality and get called from the façade!

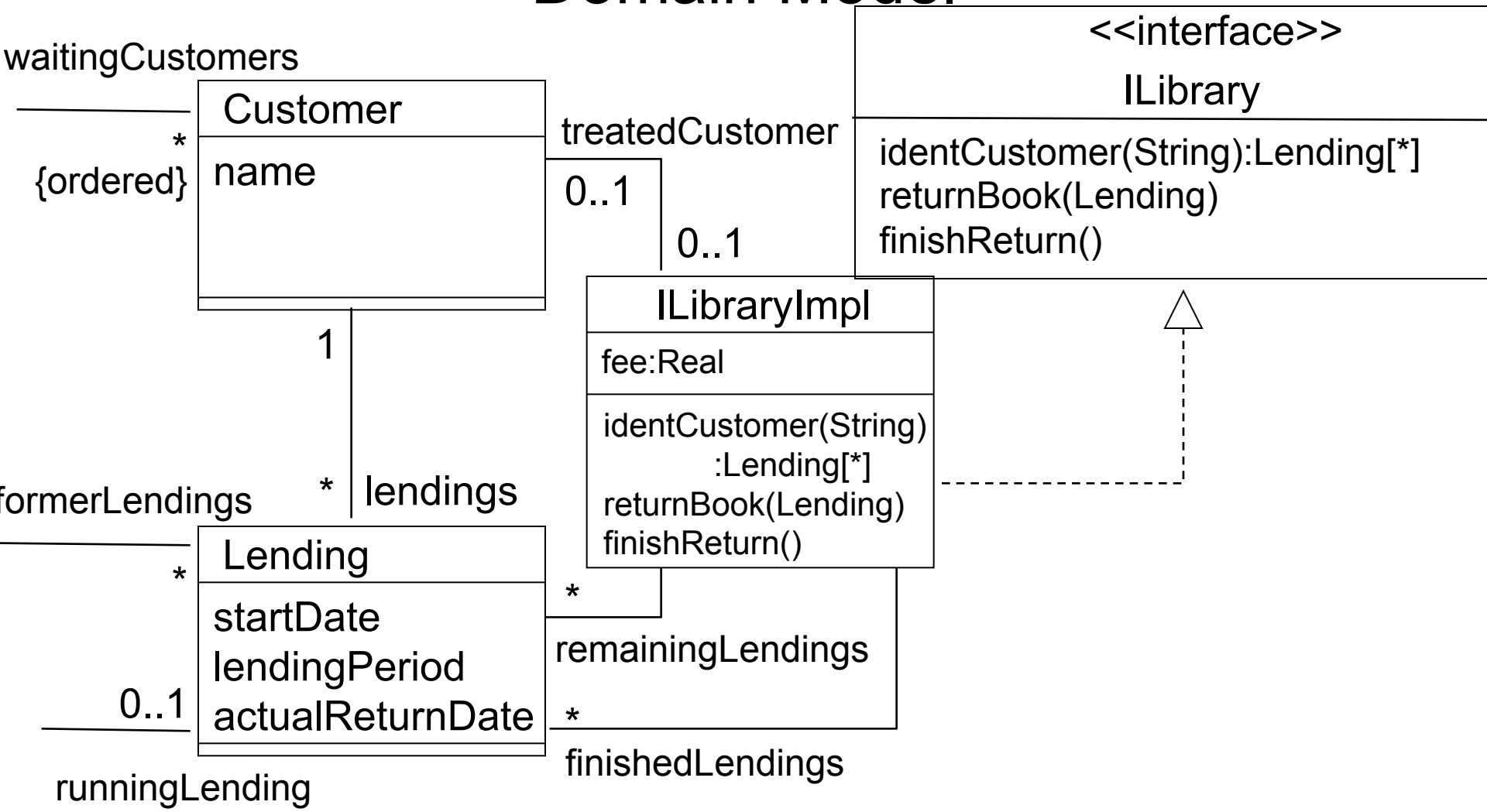
Refinement: Add an Interface



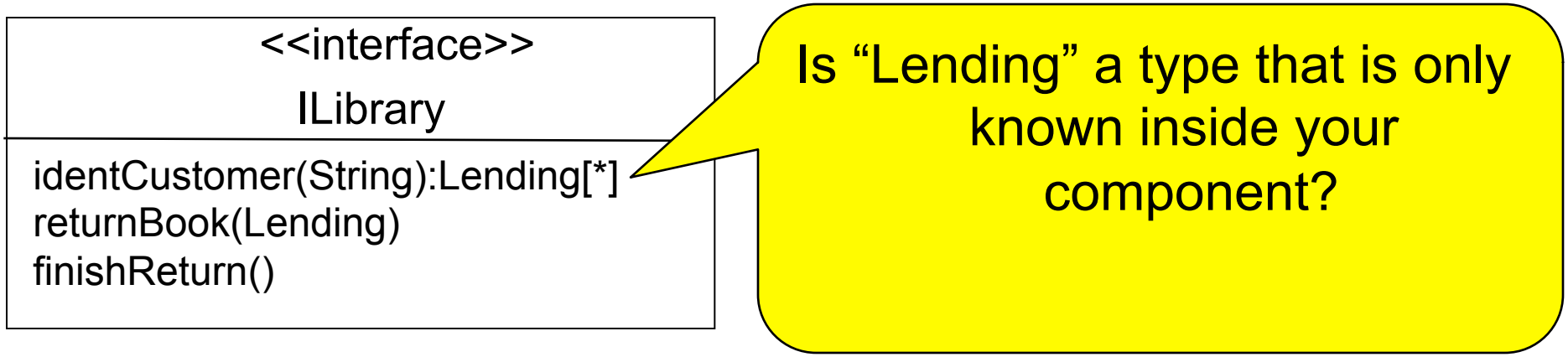
Refinement: Add Interface



Refinement: Add Interface & Facade to Domain Model



Types: What does your environment know?



- Often, the types you are using within your component (Class Diagram) are private to that component
- This means, using them as parameters for interface operations does not make sense!
- Solution: Use only primitive datatypes & collections

Learning Outcomes

- …be able to create UML Sequence Diagrams (Lifelines, different message types, different combined fragments, conditions)
- …be able to reflect on the complexity of Sequence Diagram. When is it suitable to have a higher/lower abstraction/level of detail?
- …be able to create Sequence Diagrams given component and interface definitions and/or Use Cases
- …be able to describe a process to systematically refine your system design starting from sequence diagrams with abstract components down to actual façade classes, interfaces, and classes within your component
- …be able to argument why you use/don't use certain parameter types in your component interfaces

Papyrus

- How to define Sequence Diagrams with component interactions?

→ Demo