CO2401 Software Development

Exam Revision 1

Todays aims

- Revise and discuss content likely to appear on your exam

Exam

- It will last 1.5 Hours
- There are 3 Questions
  - You will be asked to answer all 3 questions
  - 33 Marks Each
- I would like to prepare you as well as I possibly can

Exam Structure

- You will be asked to take an analytical approach to your answers
    - E.g. State advantages and disadvantages associated with different approaches/topics
  - You will be asked to apply your knowledge to unfamiliar scenarios

- Practice your writing / drawing!

- You will be asked to draw things in the exam (bring a ruler!)
  - Tables
  - Diagrams

Unlikely to be on the exam

- What I think has been well covered by your assignment
- Understanding/Interpreting Requirements
- State Transition Diagrams
- Design Level Class diagrams (not Domain Class Diagrams)
- Test Driven Development / Programming

Could Be On The Exam

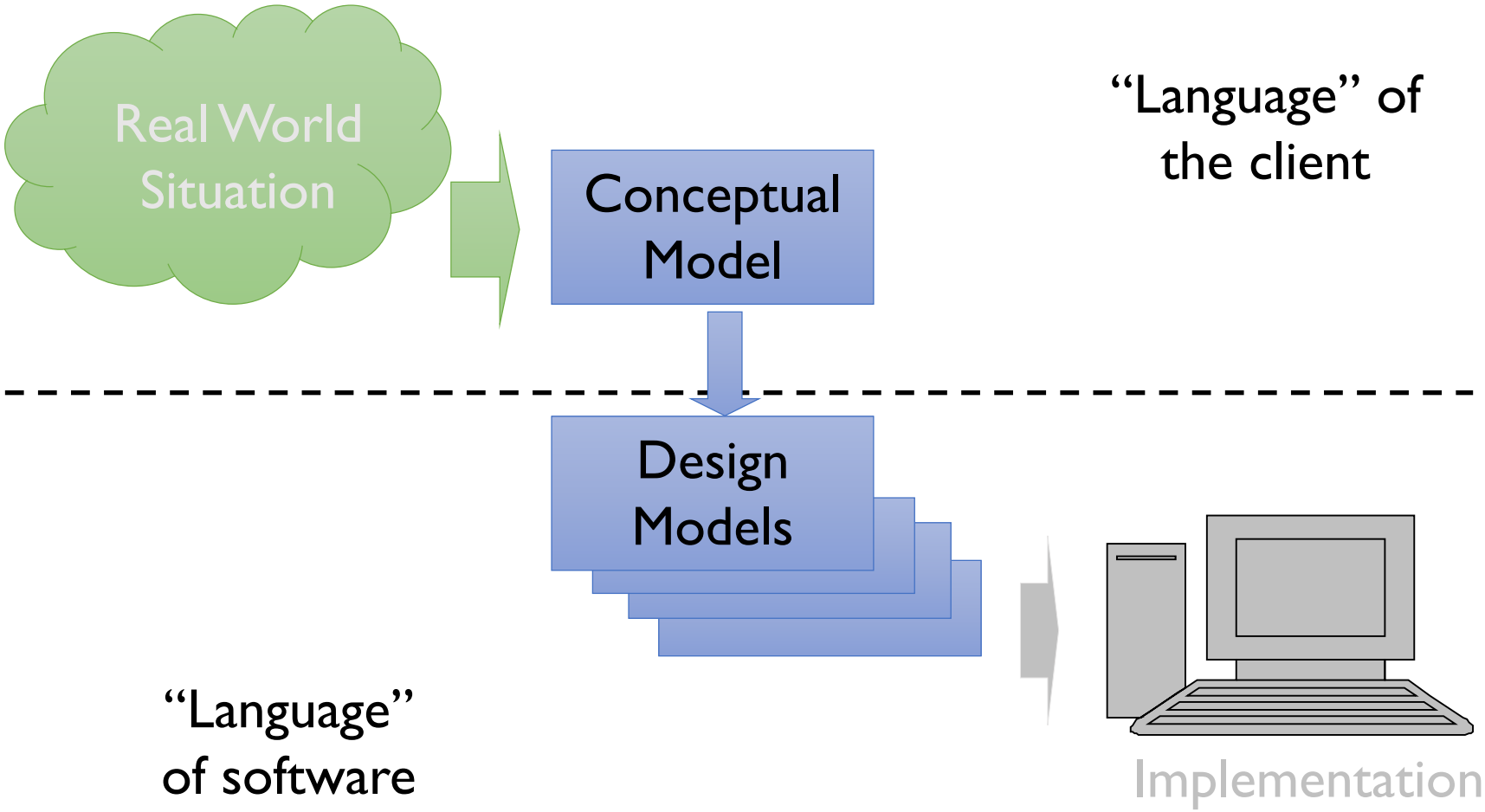
- Based on what else has been covered on the module
  - Domain Modelling
    - Noun Phrase Analysis
    - Domain Class Diagrams
  - Sequence Diagrams
  - Automated Static Analysis
  - Code Inspection
  - Test Case Design
    - Equivalence Partitions / Boundary Testing
  - Heuristic Evaluation
  - Scrum/DSDM
    - Lifecycle, Roles/Responsibilities, Rituals, Artefacts

Ask yourself: "Could I have a discussion about each of these topics"

You are unlikely to get a question about Scrum AND DSDM but should learn both.

EXAM QUESTIONS?

Conceptualising the Domain



Noun Phrase Example: A Car Hire Company

- The Car Hire Company stocks cars of a number of different models. It may have any number of cars of each model. The system should, for each model, hold the models name and engine size. No two models have the same name
- A customer can borrow any car of a particular model that is not currently out on loan. However, no customer can have more than 3 cars on loan at once. A hired car must be returned within 14 days. When a customer returns a car it immediately becomes available for loan to other customers.
- Each car has a unique Registration number
- Each customer of the hire company has a unique customer ID, the system also holds each customers name.

Noun Phrase Example: A Car Hire Company

- The Car Hire Company stocks cars of a number of different models. It may have any number of cars of each model. The system should, for each model, hold the models name and engine size. No two models have the same name
- A customer can borrow any car of a particular model that is not currently out on loan. However, no customer can have more than 3 cars and 2 models on loan at once. A hired car must be returned within 14 days. When a customer returns a car it immediately becomes available for rental to other customers.
- Each car has a unique Registration number
- Each customer of the hire company has a unique customer ID, the system also holds each customers name.

Noun Phrase Analysis of Use Cases:

- **Get Number Available:** The staff member identifies the model of the car. The system displays the number of cars available.
- **List Models:** The staff member identifies the customer. The system displays a list of the models and return dates for the cars that the member currently has out on hire.
- **Hire Car:** The staff member identifies the car and the customer who is borrowing it. The system records the fact that it is no longer available for loan, the fact that it is on loan to the member identified, and the date it was hired.
- **Return Car:** The staff member identifies the car. The system records the fact that this car is now available for loan and that it is no longer hired out to the customer who returned it.

Noun Phrase Analysis of Use Cases:

- **Get Number Available**: The staff member identifies the model of the car. The system displays the number of cars available.
- **List Models**: The staff member identifies the customer. The system displays a list of the models and return date(s) for the cars that the customer currently has out on hire.
- **Hire Car**: The staff member identifies the car and the customer who is borrowing it. The system records the fact that it is no longer available for loan, the fact that it is on loan to the member identified, and the date it was hired.
- **Return Car**: The staff member identifies the car. The system records the fact that this **car** is now available for loan and that it is no longer hired out to the customer who returned it.

Noun Phrase Example: A Car Hire Company

- Removing synonyms gives a list of candidate classes:

- Car hire company (or ‘Hire Company’)
- Car
- number of models
- number of cars
- model
- system
- Model name
- customer
- loan (or ‘rental’)
- day
- registration number
- customer number
- customer name
- staff member
- number of cars available
- list
- return date(s)
- fact
- date it was hired

Rejecting Candidate Classes

- Is it a simple property of some other candidate class?
  - E.g. title is obviously a simple attribute of film
- Is it a description of system behaviour?
  - E.g. some process or result of a process
- Is it outside the system boundary (domain)?
  - E.g. library is not within the system
- Is it part of the interface rather than the system?
- Does it describe a connection between other classes?
  - E.g. loan could relate a customer to a car, unless it has properties of its own, when it would need to be a class

Noun Phrase Example: A Car Hire Company

Is it a property of some other candidate class?

Is it a description of system behaviour?

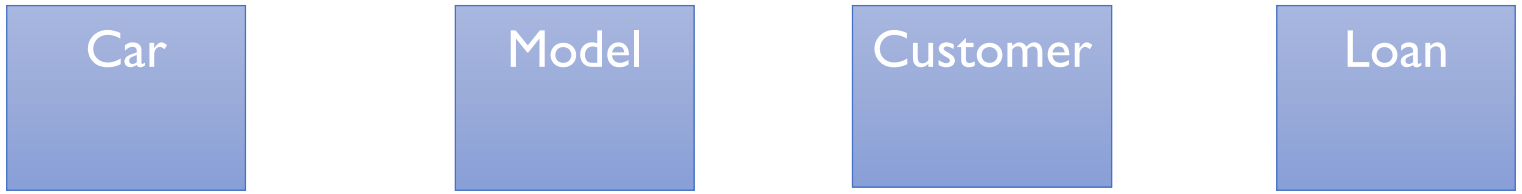
Is it outside the system boundary (domain)?

Does it describe part of the interface

Does it describe a connection between other classes?

- ~~Car hire company (or 'Hire Company')~~
- Car
- ~~number of models~~
- ~~number of cars~~
- model
- ~~system~~
- ~~Model name~~
- customer
- loan (or 'rental')
- ~~day~~
- ~~registration number~~
- ~~customer number~~
- ~~customer name~~
- ~~staff member~~
- ~~number of cars available~~
- ~~list~~
- ~~return date(s)~~
- ~~fact~~
- ~~date it was hired~~
- ~~engine size~~

Key Domain Classes



Checking Classes by Category

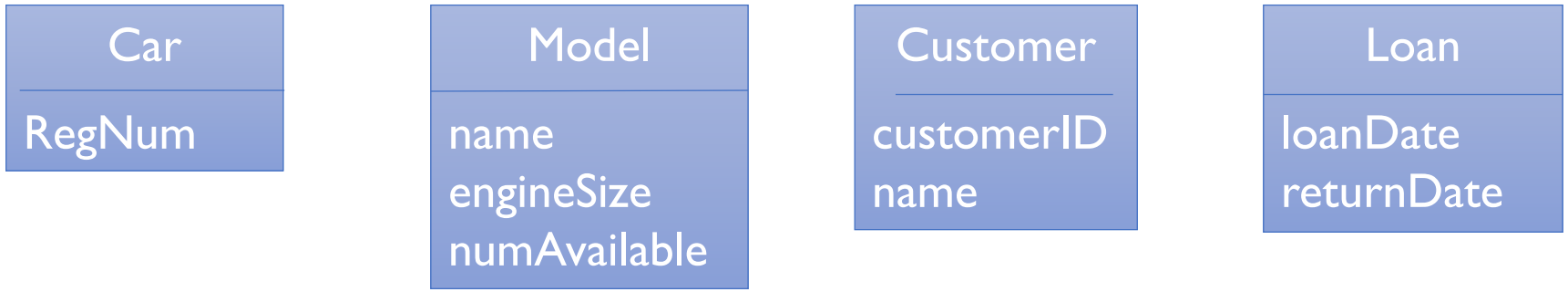
- Tangible entities
  - The physical things in the real world: can be seen or touched.
- Roles
  - played by people (and other things) in the domain
- Events
  - Any significant situation, interaction, or incident
- Organisational units
  - Groups to which people or things in the system domain belong
- Abstract entities
  - Things which exist but do not correspond to tangible entities.

Check classes identified fit into one of these categories

Identifying Attributes

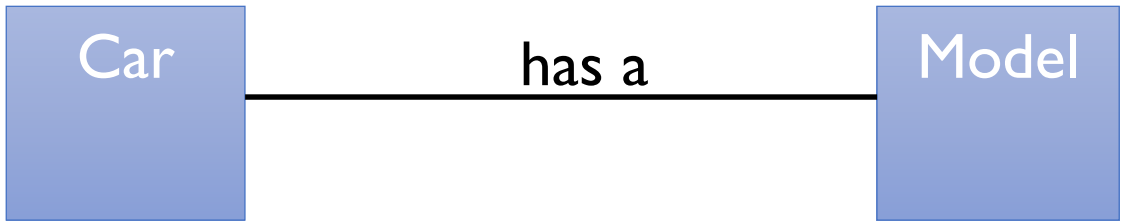
- Some of the attributes will have been identified
  - i.e. rejected as classes because they were attributes
- Whether something is a class or an attribute will often depend on the system context:
  - If further information about the “thing” other than its name or value is required, then it will have to be a class – otherwise it will be an attribute
  - E.g. the make and model of my car may be
    - an attribute in the university car-parking admin system
    - A class in my insurance company’s system (where the model may have properties such as insurance band, etc.)

Domain Classes with Attributes



Associations

- An association between two classes represents a set of links between actual objects (instances of those classes)
- Associations should be given meaningful names, but (at this stage) they are not directional
  - Just conceptual links



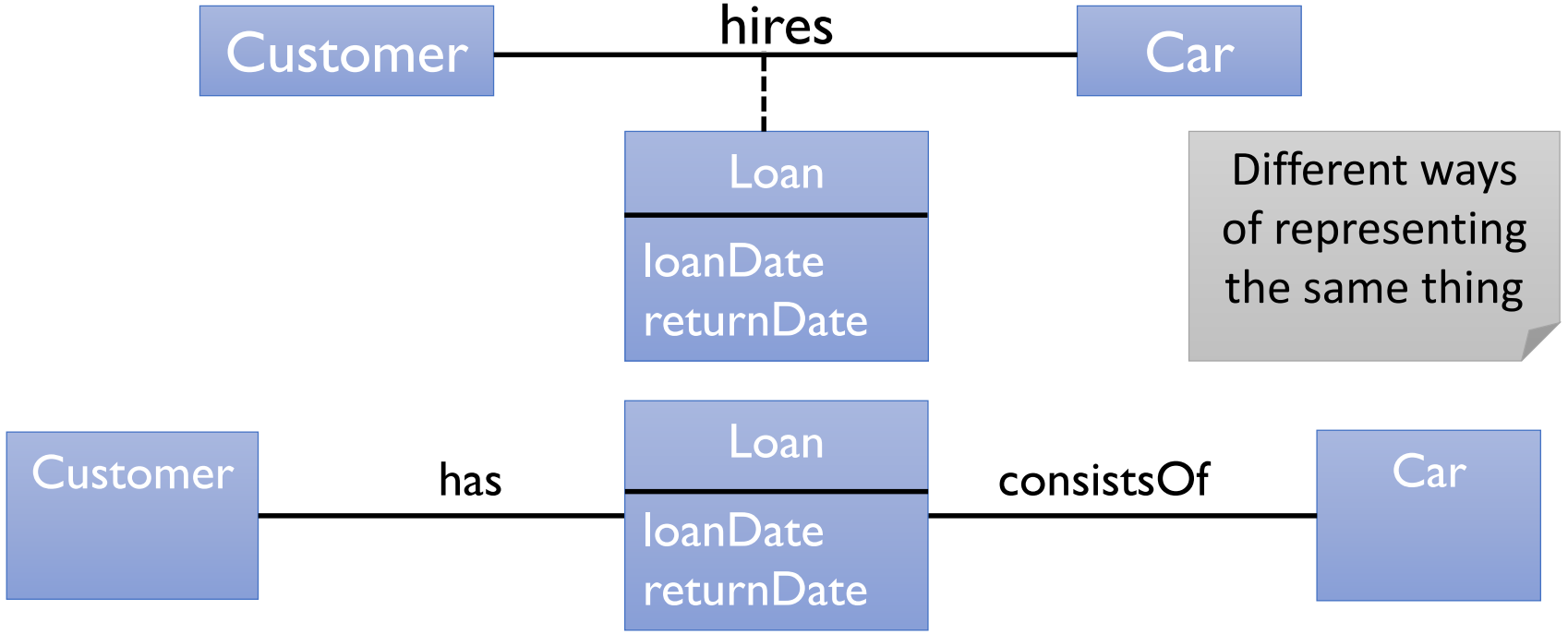
Identifying Associations

- Some associations will be obvious when classes are first identified, but study the requirements carefully to find all the associations
- Look at the way the nouns which represent classes are connected in the text – e.g.
- The Car Hire Company stocks cars of a number of different models. It may have any number of cars of each model. The system should, for each model, hold the models name and engine size. No two models have the same name
- Some relationships are not represented
  - E.g. between classes and their attributes, or things beyond the system boundary

Identifying Associations

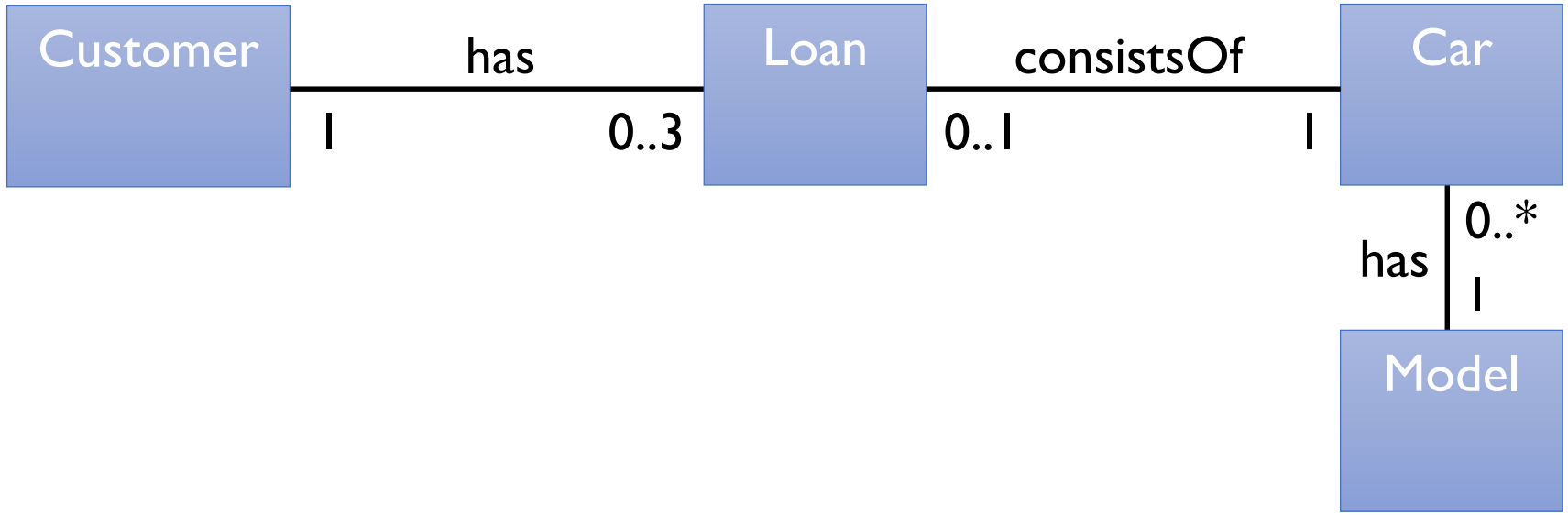
- Sometimes a relationship can have information

- There is information about the hires link (date of loan and due date) so an extra class is used:



Multiplicities

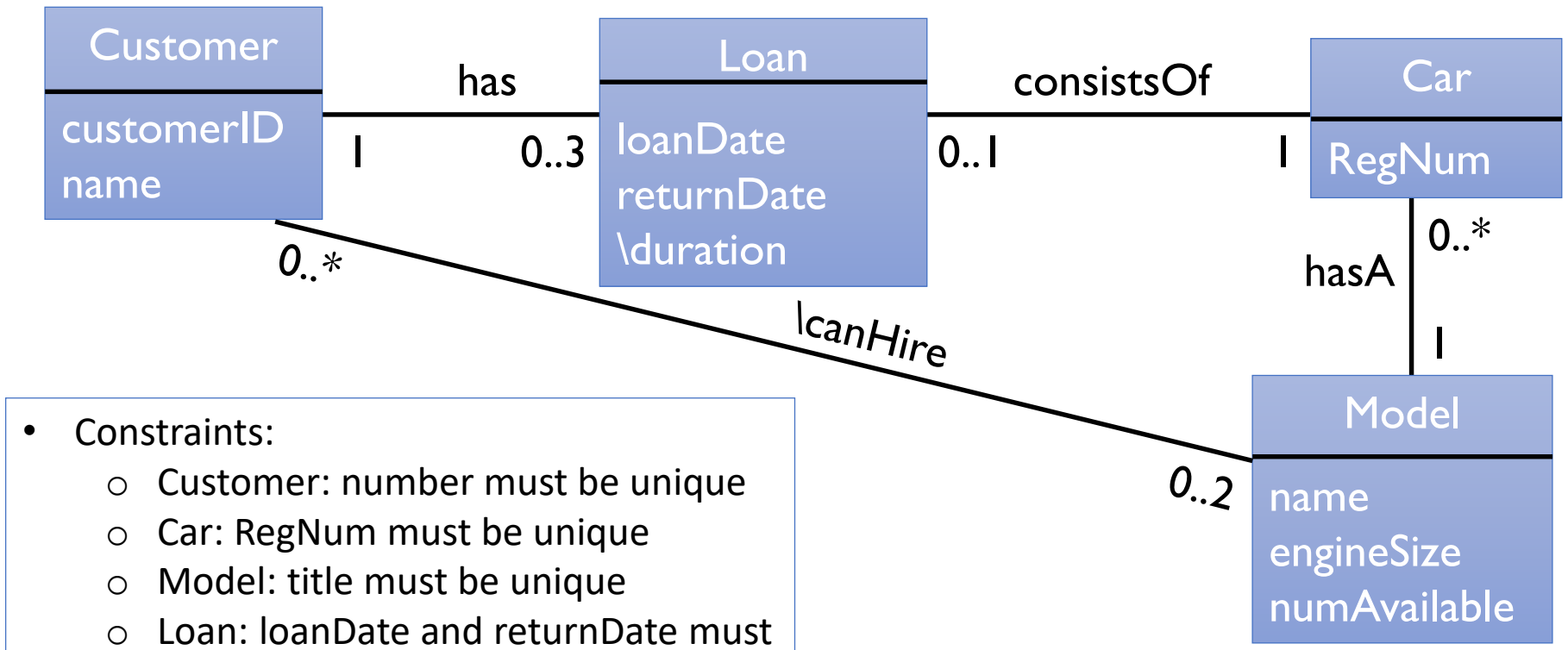
- The requirements may specify the number of individual links between specific objects
  - E.g. ‘no Customer can have more than 3 Cars on loan’.
- If important, we can show multiplicities on a class diagram



Invariants

- Invariants are constraints that must always be upheld
  - Some can be expressed graphically on the diagram
    - E.g. multiplicity constraints
  - Others refer to the values of attributes
    - On their own
      - E.g. CustomerID must be unique
    - Or when related to each other
      - E.g. loanDate and returnDate must be no more than 14 days apart
    - These invariants can be written in English as annotations on the class diagram

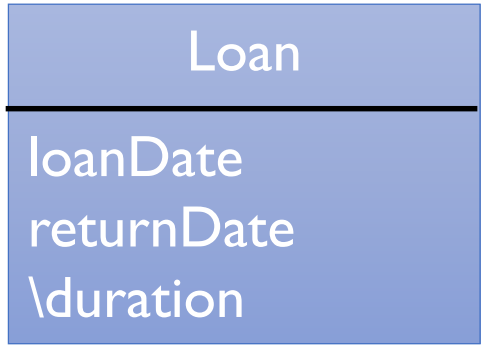
Complete Domain Class Diagram



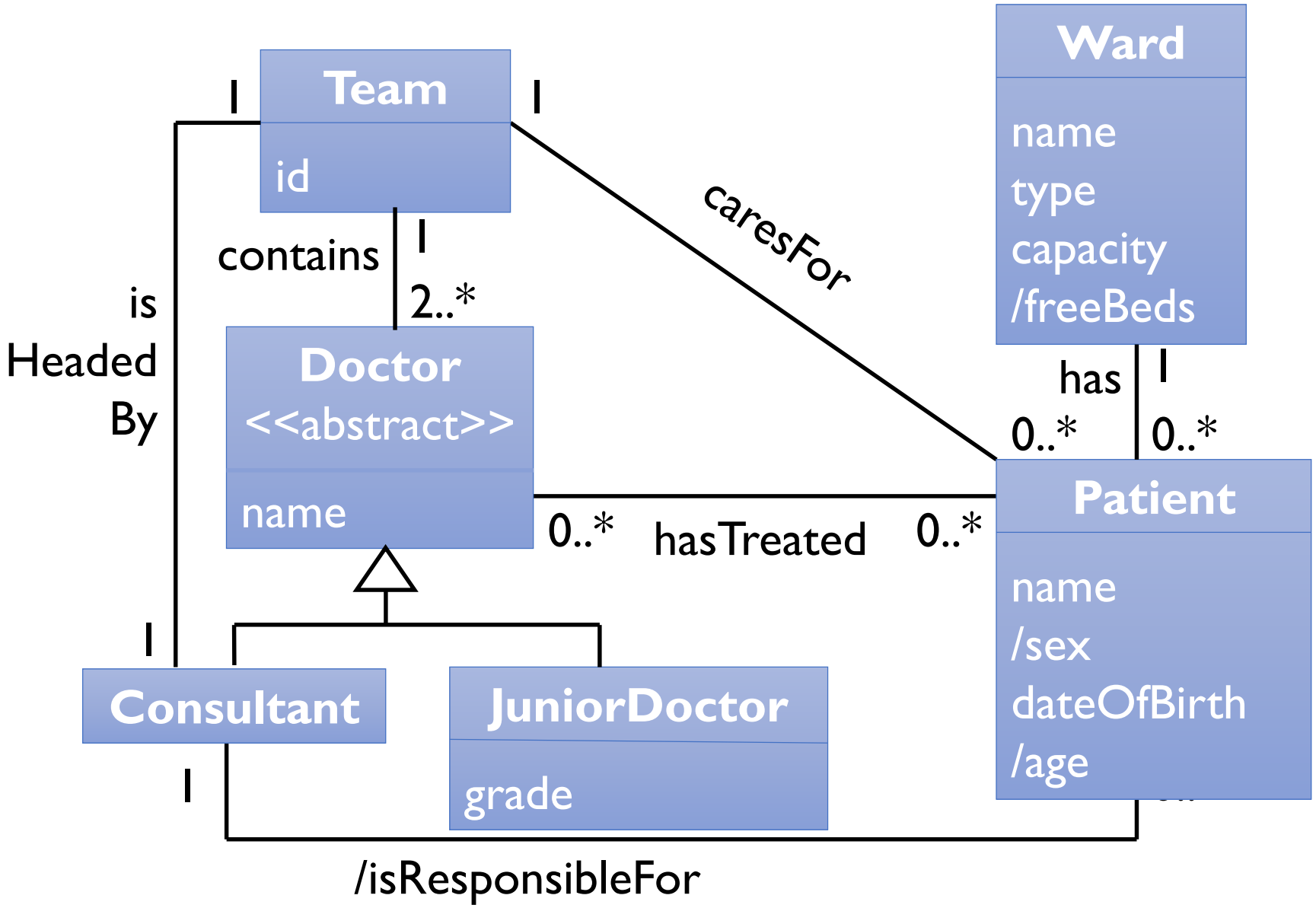
- Constraints:
  - Customer: number must be unique
  - Car: RegNum must be unique
  - Model: title must be unique
  - Loan: loanDate and returnDate must be no more than 14 days apart

Derived Attributes and Associations

- Duration can be calculated form the loan and return date
  - They should all appear on the conceptual diagram if they were mentioned in the requirements
  - But duration should be marked as a derived attribute as it may not need to be implemented as a data member
- Relationships may be derived if a path between object must exist because of invariant conditions
  - E.g. the models hired by a member
- Start derived attributes and associations with a backslash:



A Complete Hospital Solution

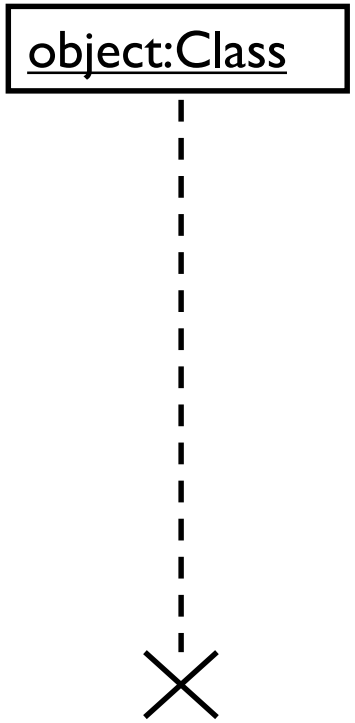


Drawing Domain Class Diagrams

- Domain class diagrams should ONLY show
  - Classes:
    - Names
    - Attributes
      - NOT visibility (Public, Private etc)
      - NOT type information
      - NOT methods
  - Relationships:
    - Names
    - Multiplicity
      - NOT role-names at the end of relationships
      - NOT arrows indicating navigability

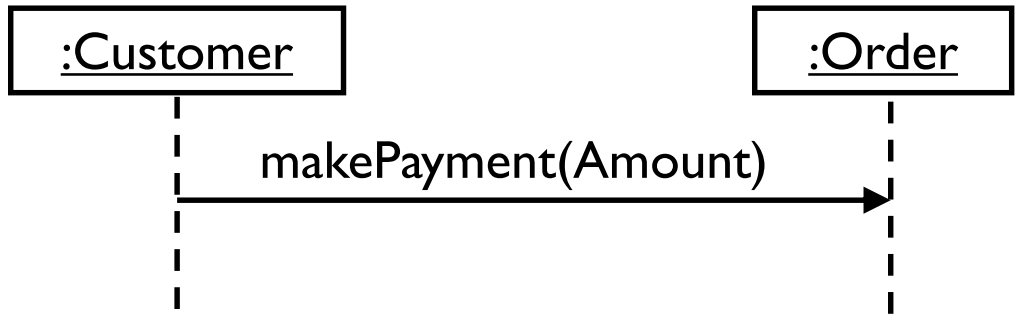
Sequence Diagram Parts: Objects

- An sequence diagram shows objects and the class they belong to.
- An object is a rectangular box at the top of its own (vertical, dashed) lifeline.
- Time moves from top to bottom
- If an object is created or destroyed during the scenario, its lifeline begins or ends at that point.
  - An object's destruction is marked with a large X.
  - Objects can self-destruct or be destroyed.



Sequence Diagram Parts: Messages

- Passes instructions & information between objects.
  - Usually an object calling a method of another object
  - Represented by labelled arrows between object lifelines



- Basic features of messages are:
  - Data content (the parameter values)
  - Synchronisation pattern
    - Stop & wait (synchronous)
    - Carry on in parallel (asynchronous)

Sending a message with data is like **calling** a method with parameters

Synchronisation

- Synchronous message passing (filled arrowhead)
  - The sender is blocked from continuing until the target completes its processing of the message

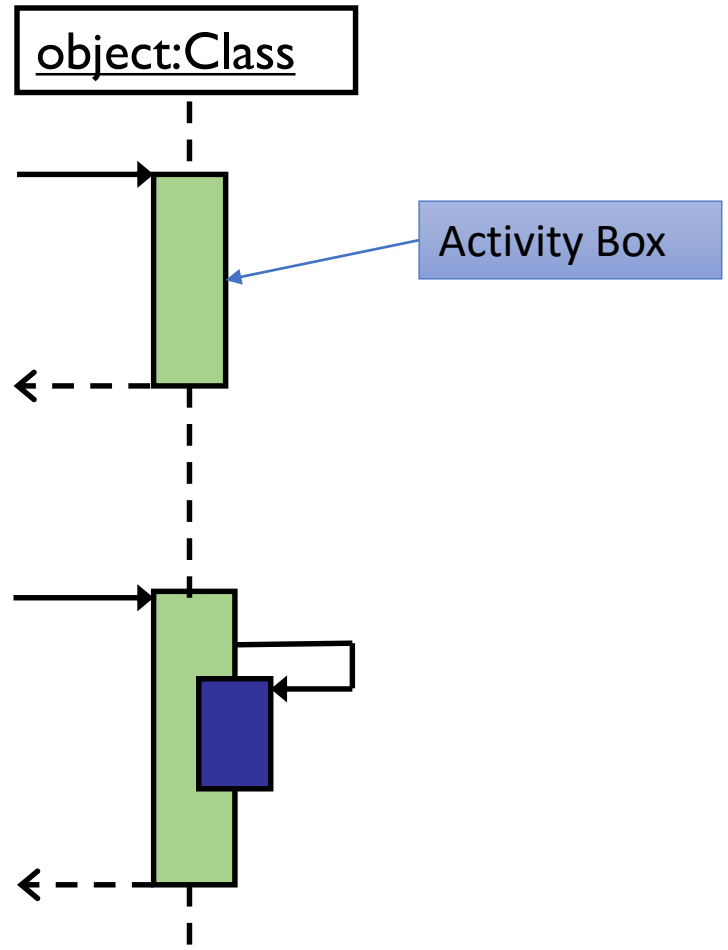
    - Example: method call in a single threaded program
- Asynchronous message passing (stick arrowhead)
  - Caller isn't blocked, so it carries on with its own processing

    - Example: sending an email
- Return from a call (dashed line, stick arrowhead)
  - Only show return if necessary

- There are other synchronisation patterns.

Object Activation

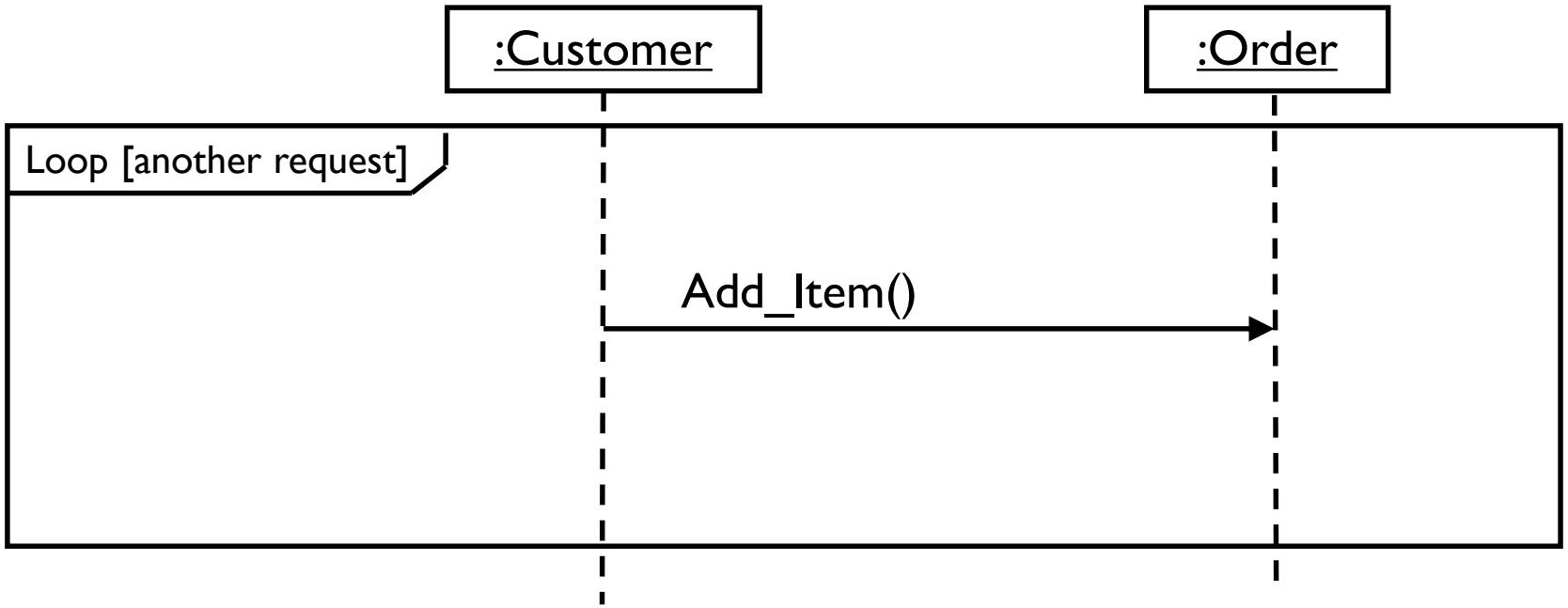
- An object is activated when doing an action.
  - i.e. a method of the object is executing
  - Shown as a thin rectangle, an activity box.
- If an activated object calls itself, a second activity box is drawn slightly to the right, so it "stacks up" visually.
  - Such calls to one of the object's own methods can be nested
    - Call further methods



Interaction Frames

- Sequence diagrams may contain “interaction frames” (boxes) to group several interactions together
  - **opt** (“optional”):
    - region will only be carried out if guard is true
  - **alt** (“alternative”)
    - Do at most one alternative, if a guard (condition) is true
    - can have an “else” part
  - **loop** (repetition)
    - Can specify min/max number of repetitions or a Boolean guard like a while-loop condition, e.g. loop [anotherRequest]
  - **ref** (reference)
    - Refers to an interaction frame defined somewhere else

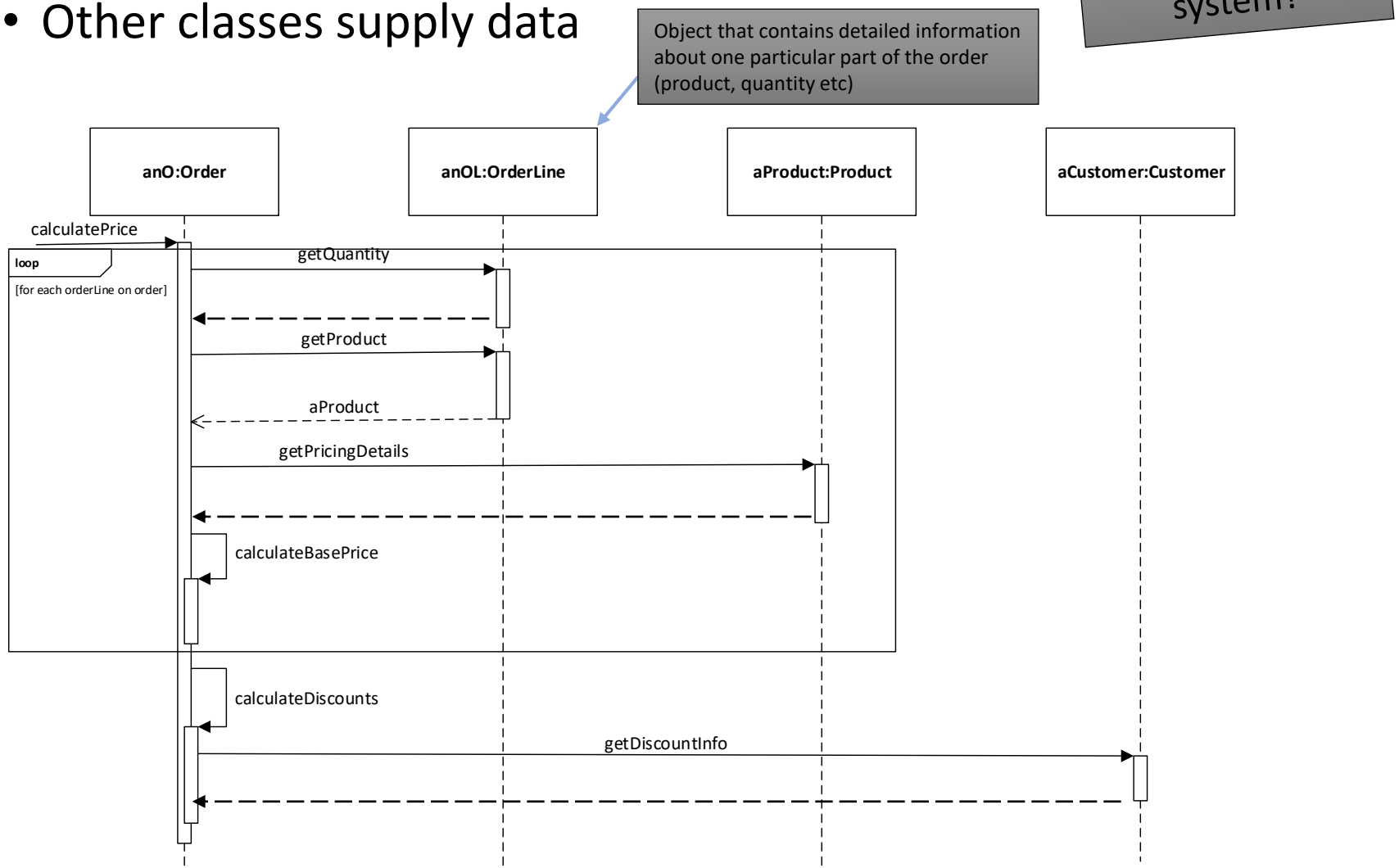
Example Interaction Frame (loop)



Example 1 centralised control

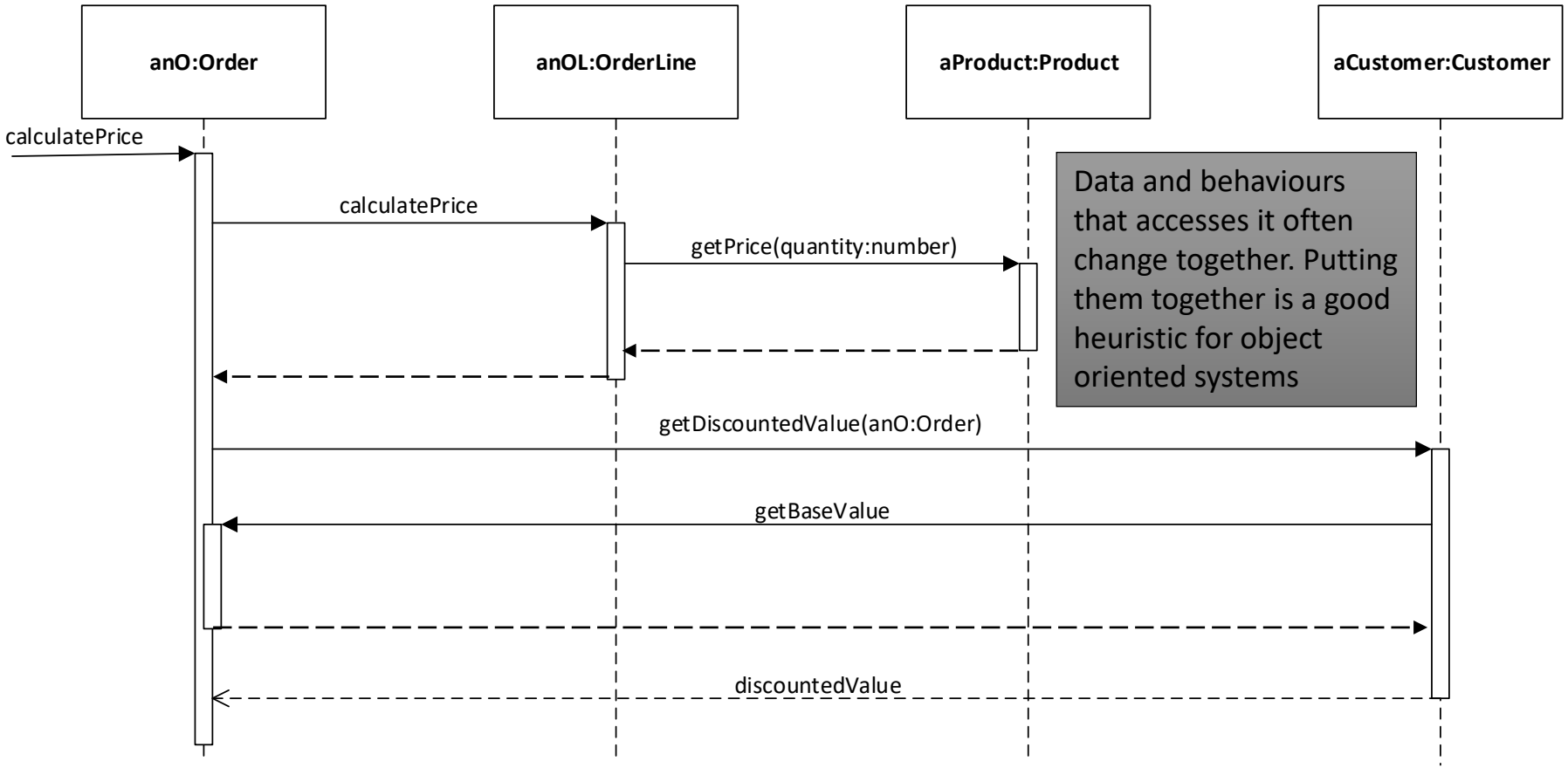
- Order doing most of the processing
  - Other classes supply data

Is this a good design for an OO system?



Example 2 Decentralised control

- Processing split among many classes each implementing a part of the algorithm
  - Data and associated behaviours grouped together



Model Dependencies

- Various diagrams are all inter-related
- Remember that the development process is iterative:
  - Adding details to one diagram might require us to go back and update other diagrams or descriptions as we discover more about the nature of the system