System Analysis and Design

Object Oriented Analysis

Salahaddin University
College of Engineering
Software Engineering Department
2011-2012

Amanj Sherwany
http://www.amanj.me/wiki/doku.php?id=teaching:su:system_analysis_and_design

Object-Oriented Analysis

- The am is identifying:
  - the objects in the system
  - their properties and/or interconnections
  - their behaviours and/or responsibilities
  - and potentially also groupings of objects into classes and inter-class relationships with respect to generalisation
- We can analyse an *existing system* to try to derive these properties (too late), **or** we can be driven by the *requirements* in a more constructive fashion.

Object-Oriented Analysis, *Cont'd*

- OO analysis comes after requirements elicitation, not as a part of.
- **Object-Oriented Analysis:** is the use of object modelling for functional requirements.
- OO analysis is the first step of the design process to develop an overview of the system and its important components.

Analysis for Design

- Without proper analysis, a design is likely to be wrong which can be devastating if not timely detected.
- Common analysis activity will generally produce a number of:
  - use case scenarios
  - conceptual models or “class diagrams”

Use Case Scenarios

- A use case should describe what the system shall do for the user (or *actor* in UML terminology) to achieve a particular goal at an appropriate level or detail without any implementation specifics.
- Each use case should constitute a “complete course of events” from the actors' point of view.
- Different actors are used to model different roles that users may have when interacting with the system.

Use Case Scenarios, *Cont'd*

- Examples of suitable actors for an OS might be 'user', 'guest', and 'super user'
- The advantage of visual use case description is that they are easy to read since flow is more easily expressed in diagrammatic notation rather than in text.
- Use cases are excellent starting points for building system tests.

Textual Use Case: Transfer Money

Name: Transfer money

Purpose: Allows the actor transfer money between accounts

Optimistic flow:

1. Actor logs into the system
2. Actor selects *from* account, enters *to* account and a *sum*
   a. If the sum is ≥ 200 USD, a fee of 2% the sum is added
   b. If the sum is < 200 USD, no fee is added
3. The update balance(s) is displayed

Pessimistic flow:

Problem 1: No *from* or *to* account selected/entered

1. Actor is prompted to select the *from* account/enter *to* account

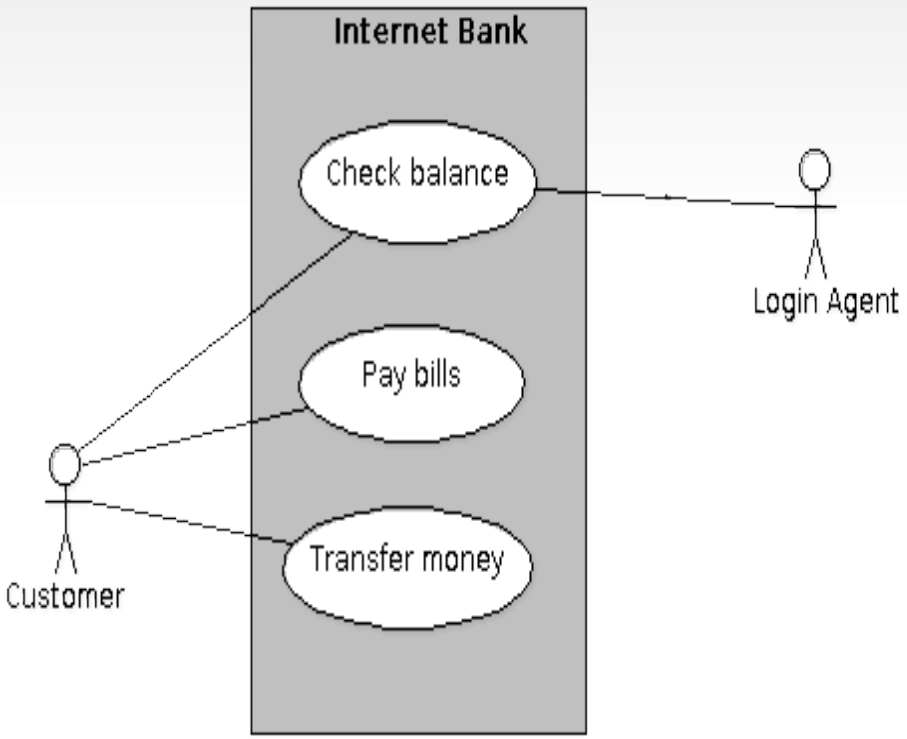
Problem 2: *To* account does not exist

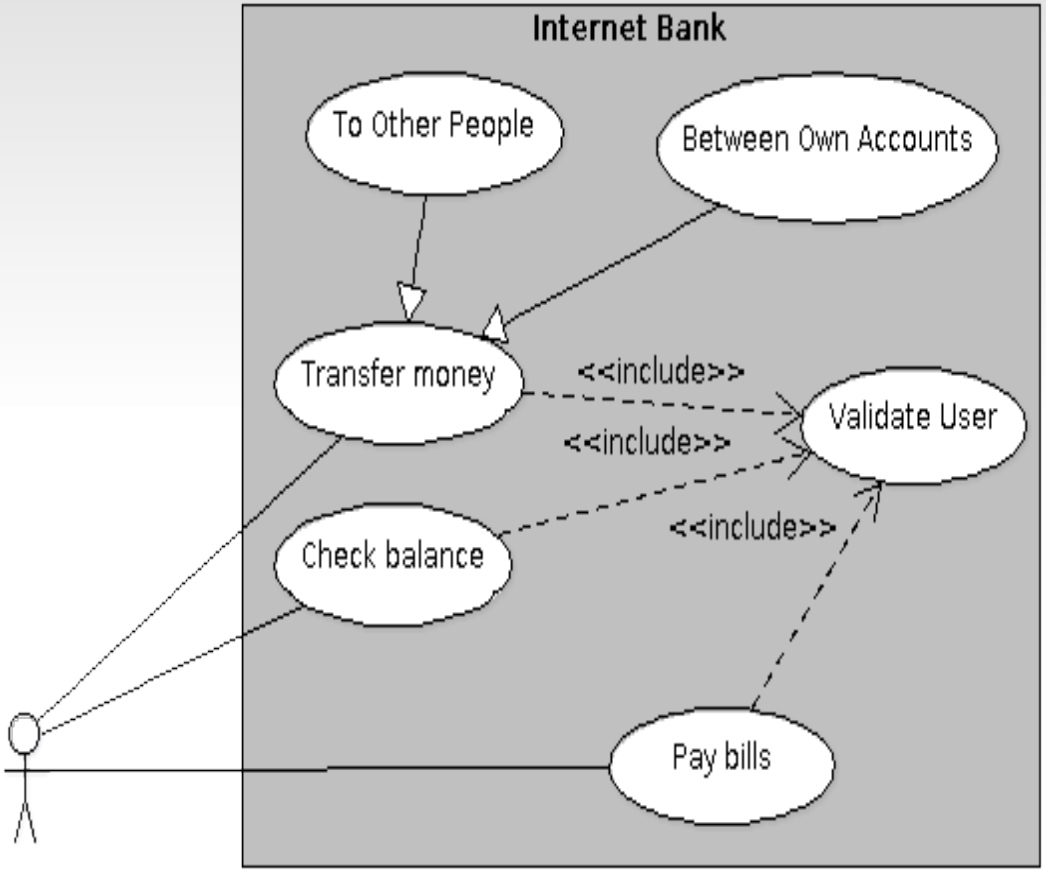
1. Actor is notified that the *to* account does not exist
2. The current (unchanged) balance is displayed

Problem 3: *Sum* exceeds available funds

1. ….

Use Case Diagrams: Transfer Money





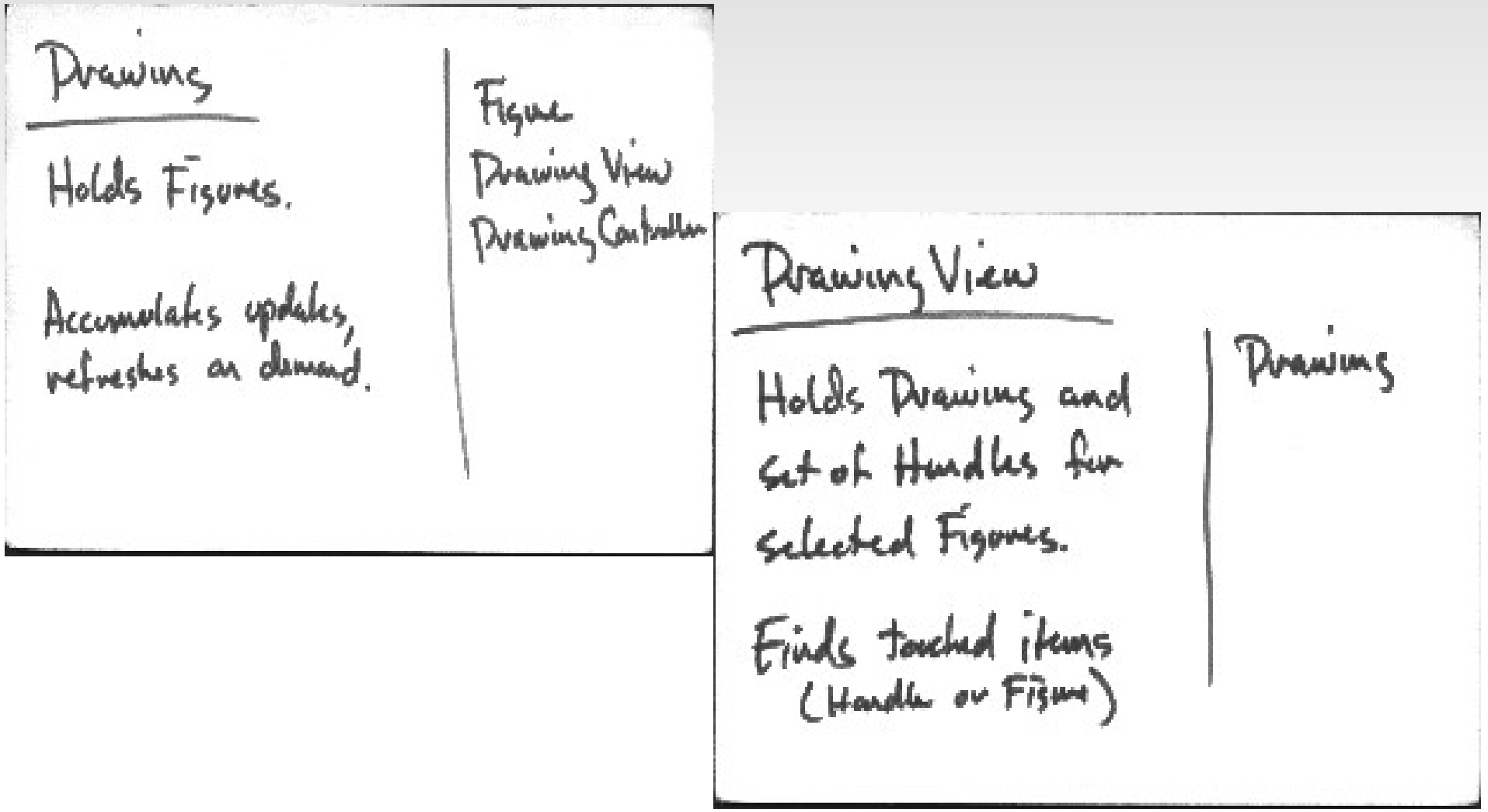
Analysis Techniques for Identifying Classes

- There is no good way to identify classes properly, even formal models are not that good.
- There are some “decent” ways, though:
  - Analysis with CRC Cards
  - Analysis of Natural Language

Analysis with CRC Cards

- CRC stands for Class, Responsibility, Collaborators
- CRC cards are normal index cards –one card per class
- Analysts write the *class' name*, its *responsibilities* in the system, and what classes it is *collaborating* with to fulfil its responsibilities
- CRC cards can be filled out in a way similar to use cases where team members walk through system scenarios

Analysis with CRC Cards, *Cont'd*



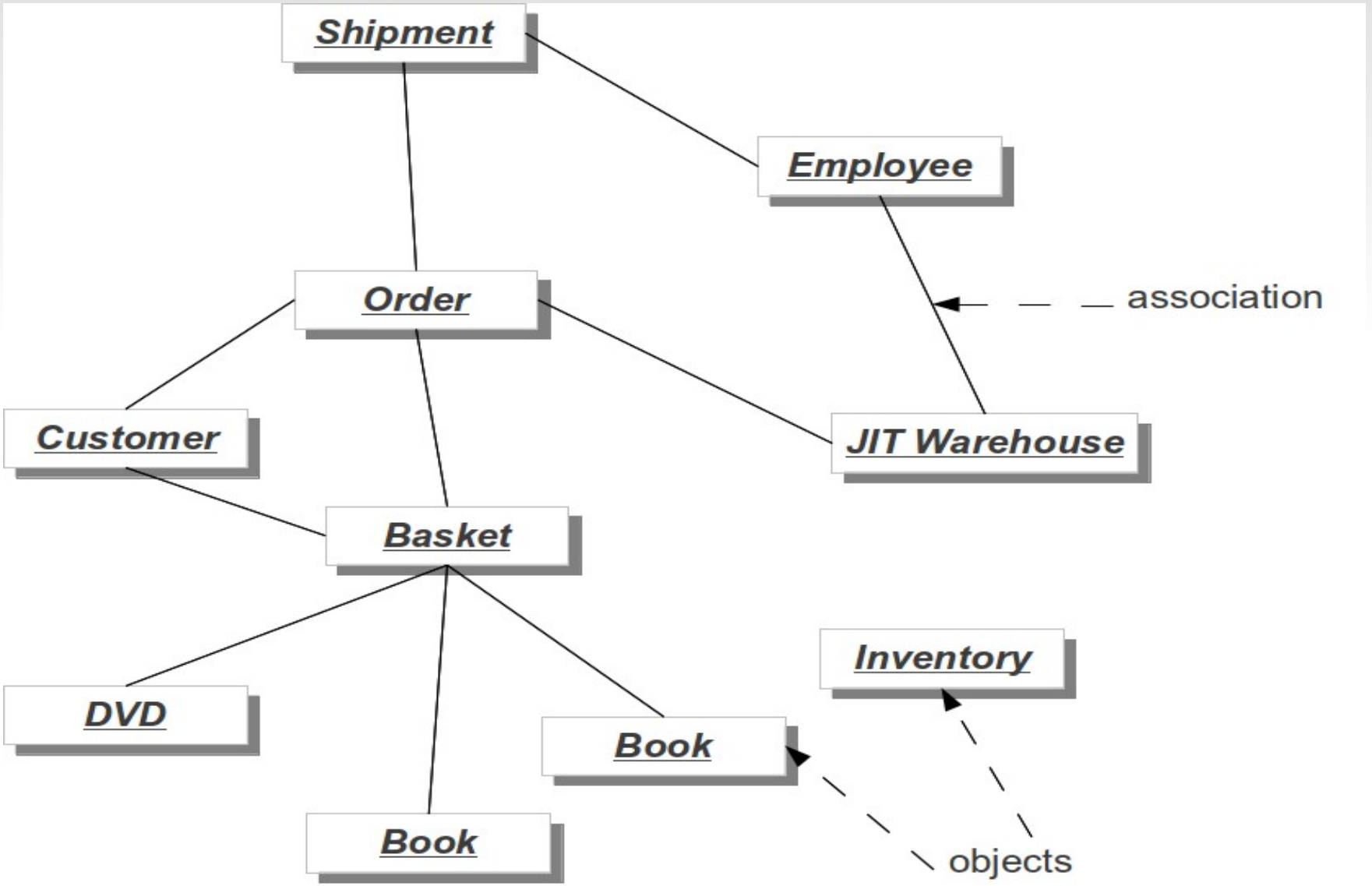
Analysis of Natural Language

- In this approach we look at the written requirements and analyse the language use.
- This is not entirely unproblematic, as the writer and writing style clearly influences the wording of the requirements.
- However this technique for identifying *possible* actors and operations is still very useful.
  - Perhaps even before proceeding with a CRC analysis

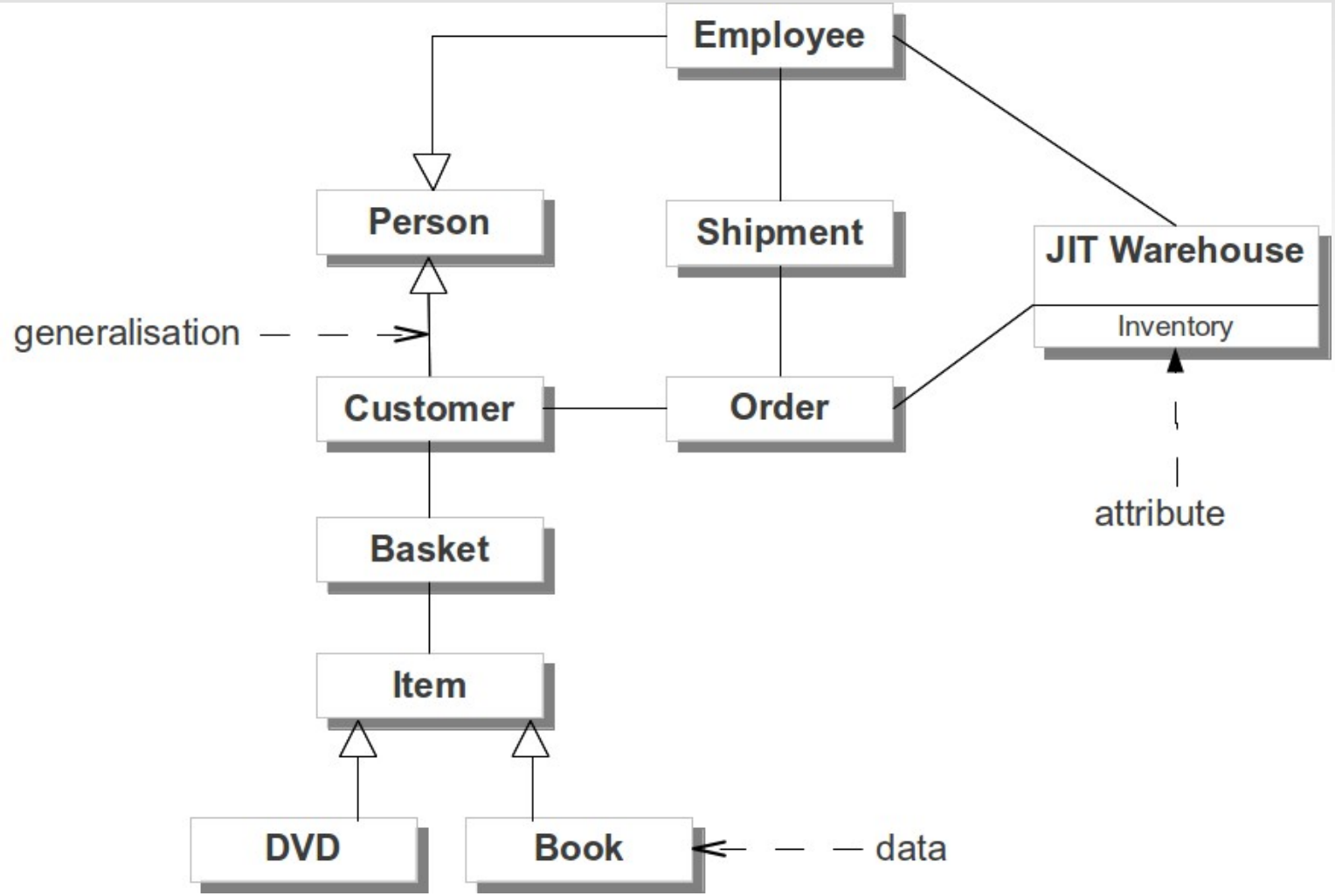
Class Diagrams

- **Class diagrams:** capture the classes in a system, and their interconnections.
- They can be drawn at many levels of detail, and it might be useful to draw several different class diagrams for the same system, with different foci.
- A class diagram might perhaps start with a simple object diagram, or it might as well start with the results of a CRC analysis.
- At this level, we might even introduce some generalisations.

Object Diagram



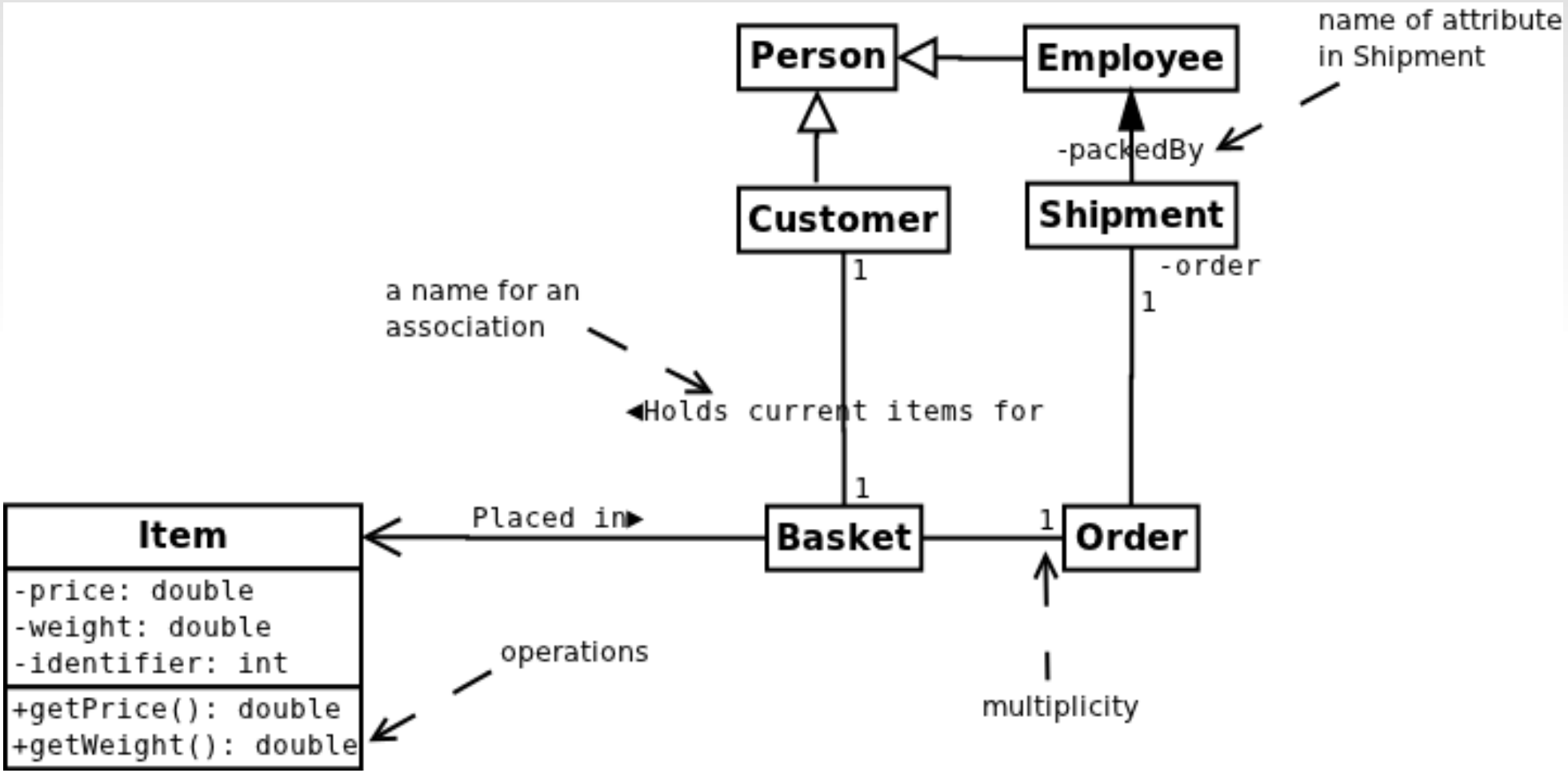
Class Diagram



Developing Class Diagrams

- A next step in developing the previous class diagram could be to add:
  - *Multiplicities*
  - *Roles or names*
  - *Perhaps followed by attributes and operations*

Developing Class Diagrams, *Cont'd*



Developing Class Diagrams, *Cont'd*

- The notation (-) stands for private and (+) for public
- Directions are added, where necessary to point out. For example, there seems not to be possible to *navigate* from employee to the shipments she has packed, nor from items to baskets.
- Without explicit directions, associations are understood to be bidirectional, which is seldom the case in practise in an implementation.

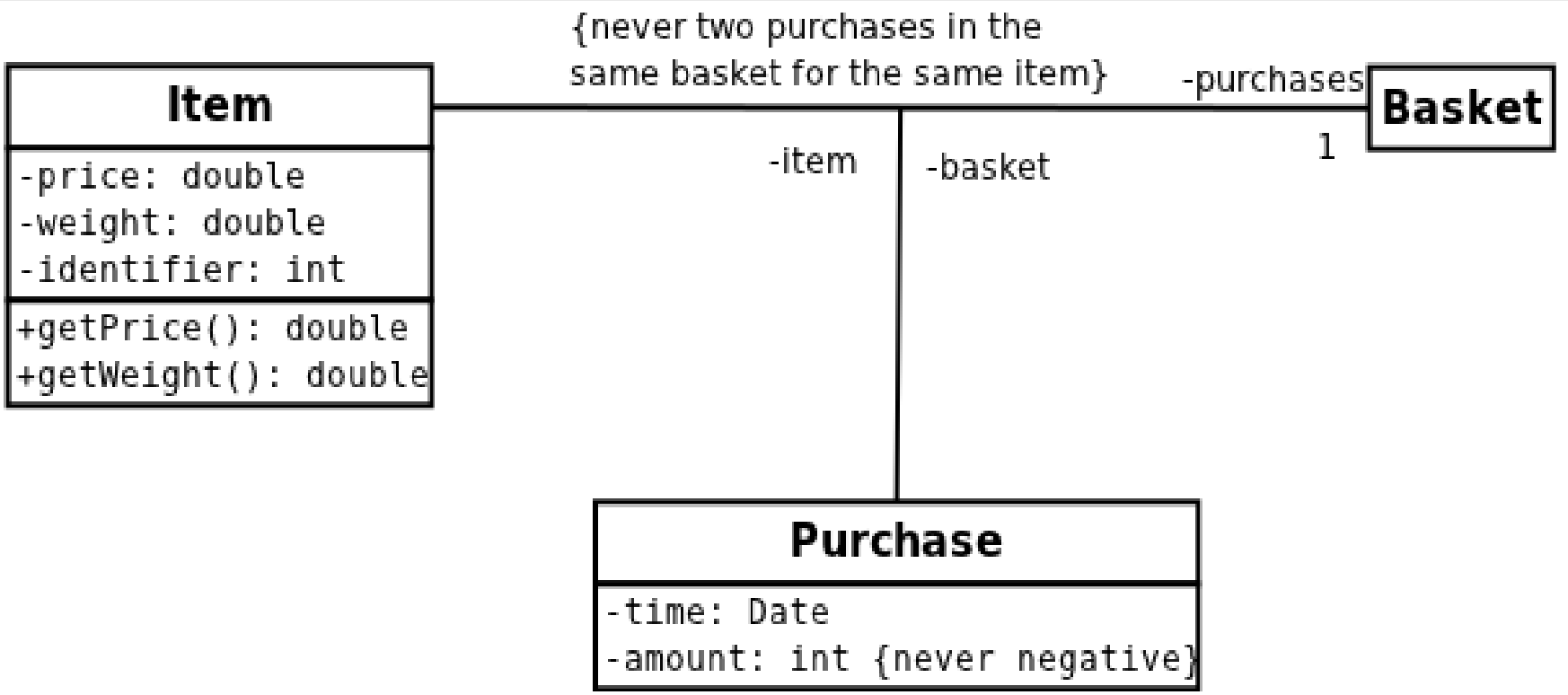
Developing Class Diagrams, *Cont'd*

- Sometimes, an association has interesting semantic properties that we may want to capture.
- For example, maybe we wish to allow purchases of several copies of a book, and maybe time-stamping when the books are placed in the basket to resolve prioritising when running out of stock
- We could represent it like the follow diagram.

Developing Class Diagrams, *Cont'd*

- Further down the road, we might wish to capture constraints in the class diagram.
- At this point, we will be content with simple natural language statements written in { and }, which is a standard UML syntax.

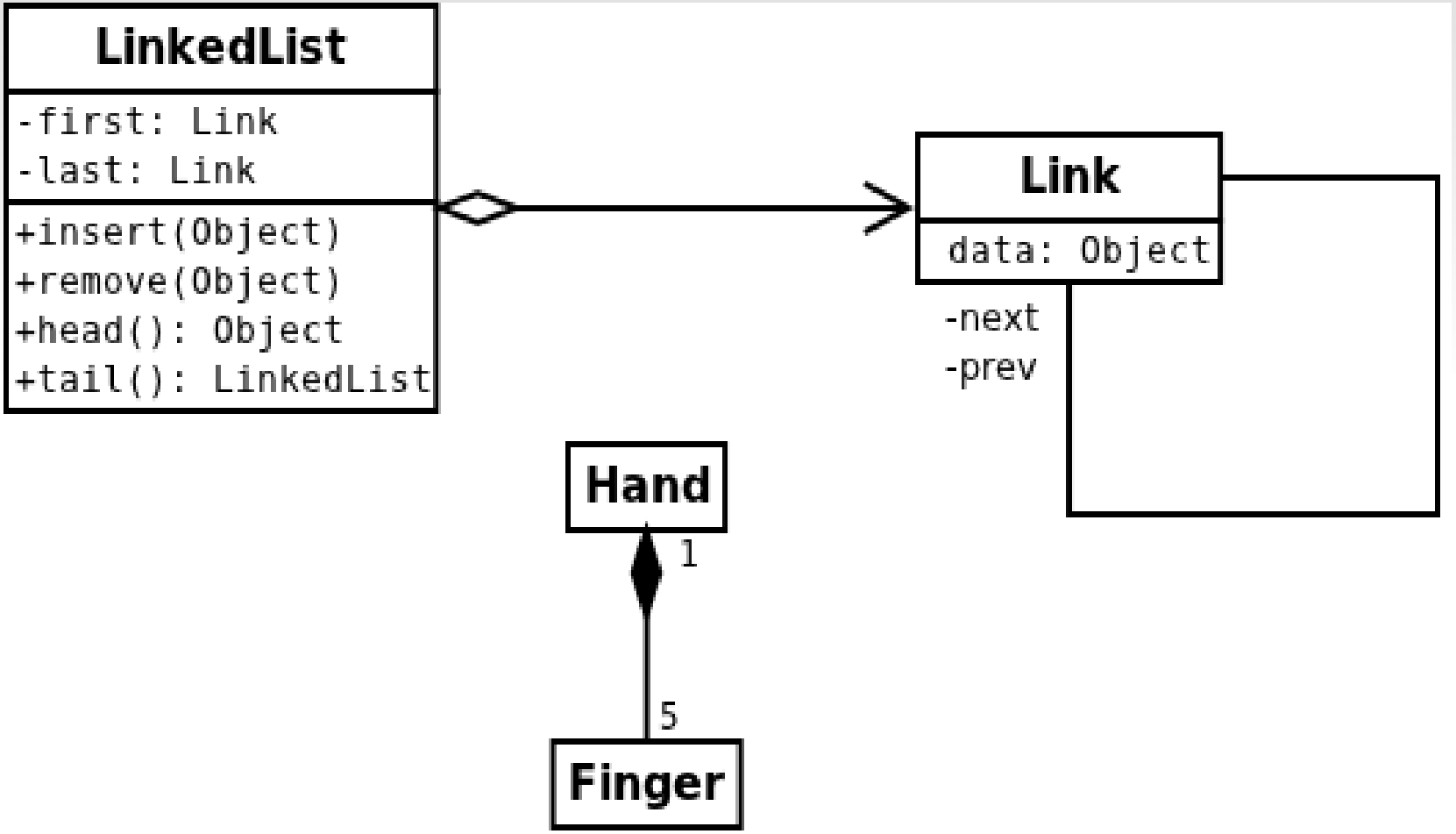
Developing Class Diagrams, *Cont'd*



Aggregation and Composition

- Objects are generally quite small and useless on their own
- Rather, objects are combined into aggregates, or aggregate objects.
- In UML *Aggregation* is denoted by a white diamond.
- A black diamond is used to denote *Composition* in UML.

Aggregation and Composition, *Cont'd*



Next?

- Use case diagrams (and CRC card stacks) are used as a basis for *sequence diagrams*.
- Sequence diagrams mode the behaviour of a use case, emphasising the time-based flow of event.