Refining the Requirements Model

Based on Chapter 8 of Bennett, McRobb and Farmer:

Object Oriented Systems Analysis and Design Using UML, (4th Edition), McGraw Hill, 2010.

Mc Graw Hill Education

In This Lecture You Will Learn:

- About reuse in software development;
- How OO contributes to reuse;
- How to identify and model aggregation, composition and generalization;
- An approach to modelling components;
- About ‘patterns’ in software development;
- How analysis patterns help to structure a model.

Reuse in Software Development

- Software development has concentrated on inventing new solutions
- Recently, the emphasis has shifted
- Much software is now assembled from components that already exist
- Component reuse can save money, time and effort

Reuse in Software Development

- Achieving reuse is still hard
  - Reuse is not always appropriate – can't assume an existing component meets a new need
  - Poor model organisation makes it hard to identify suitable components
  - The NIH (Not-Invented-Here) syndrome
  - Requirements and designs are more difficult to reuse than code

Reuse: The Contribution of OO

- Generalization allows the creation of new specialised classes when needed
- Encapsulation makes components easier to use in systems for which they were not originally designed
- Aggregation and composition can be used to encapsulate components

Adding Generalization Structure

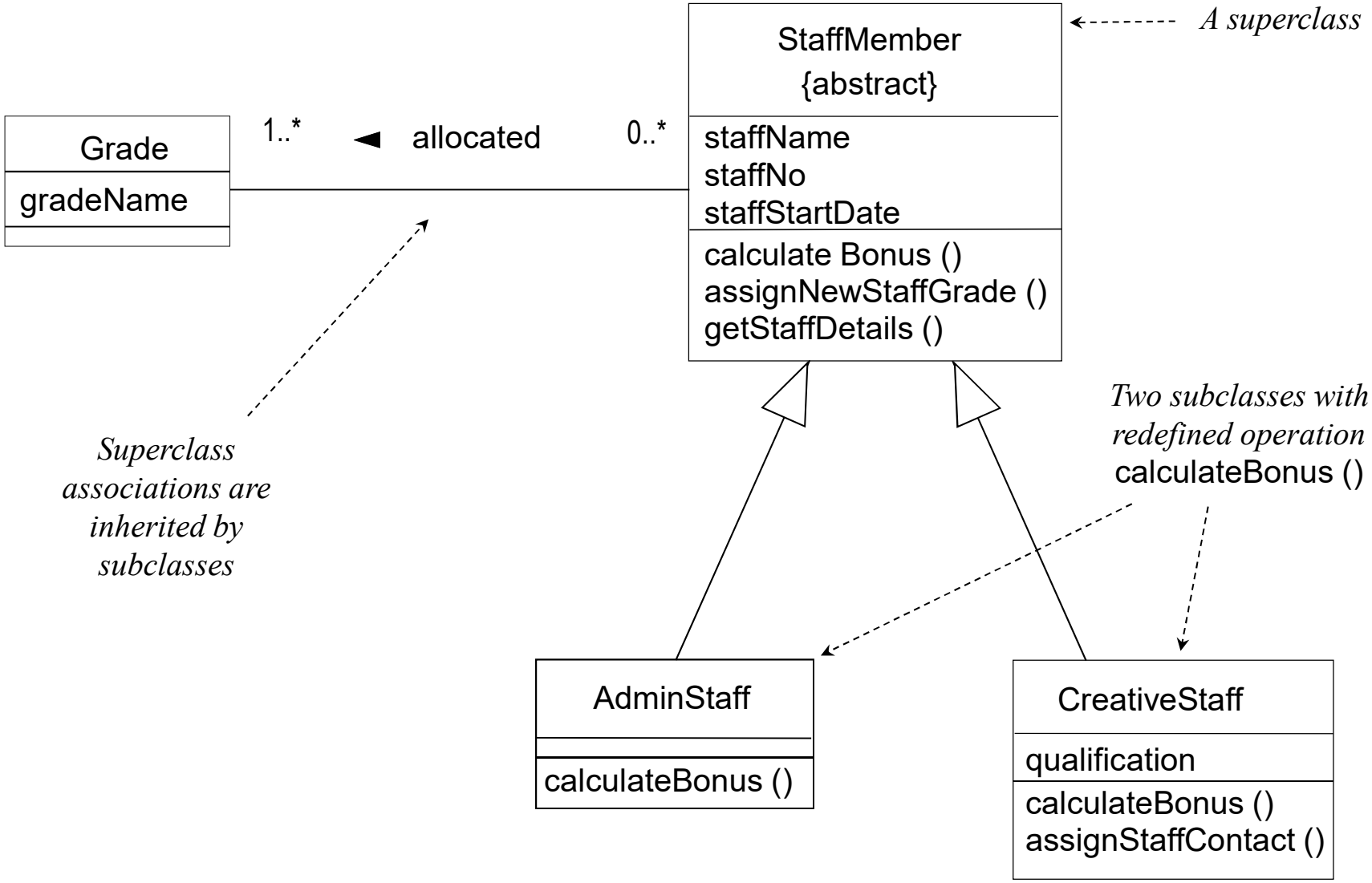
- Add generalization structures when:
  - Two classes are similar in most details, but differ in some respects
  - May differ:
    - In behaviour (operations or methods)
    - In data (attributes)
    - In associations with other classes

Adding Structure

- Two types of staff:

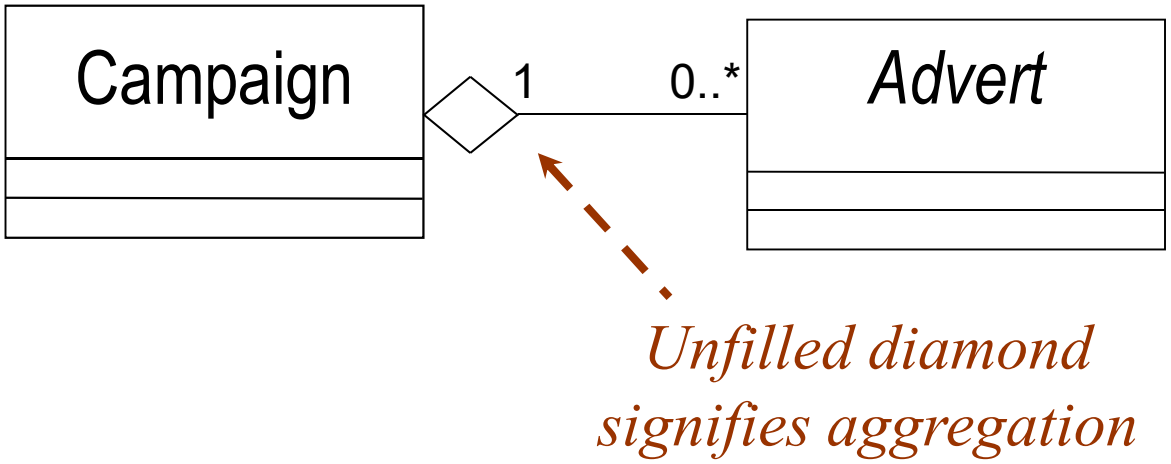
Creative	Have qualifications recorded Can be client contact for campaign Bonus based on campaigns they have worked on
Admin	Qualifications are not recorded Not associated with campaigns Bonus not based on campaign profits

Adding Structure:



Aggregation and Composition

- Special types of association, both sometimes called whole-part
- A campaign is made up of adverts:

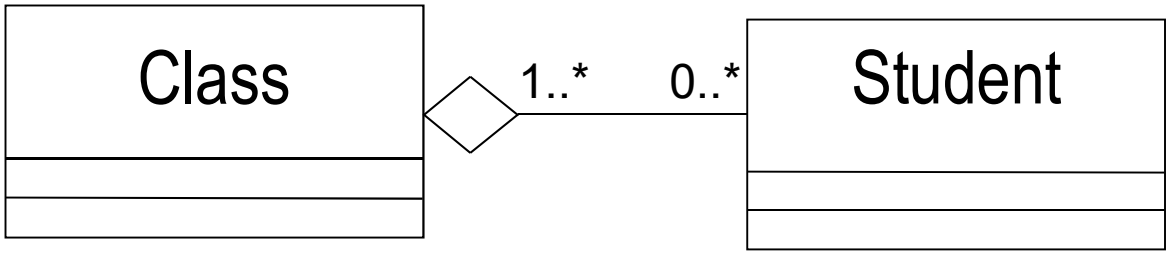


Aggregation and Composition

- Aggregation is essentially any whole-part relationship
- Semantics can be very imprecise
- Composition is ‘stronger’:
  - Each part may belong to only one whole at a time
  - When the whole is destroyed, so are all its parts

Aggregation and Composition

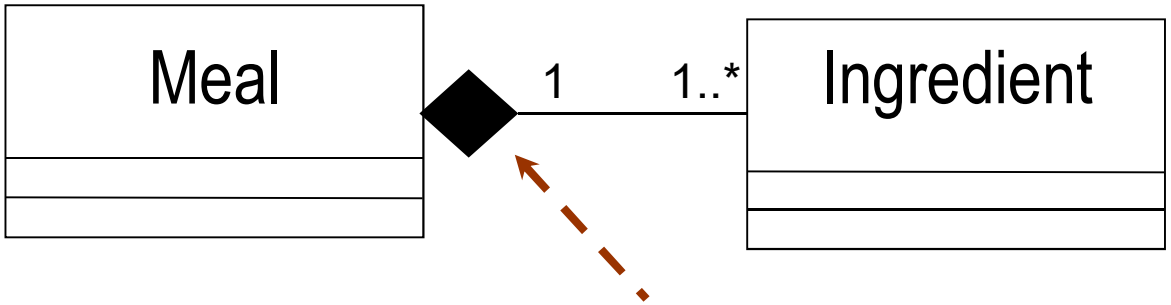
- An everyday example:



- Clearly not composition:
  - Students could be in several classes
  - If class is cancelled, students are not destroyed!

Aggregation and Composition

- Another everyday example:



Filled diamond signifies composition

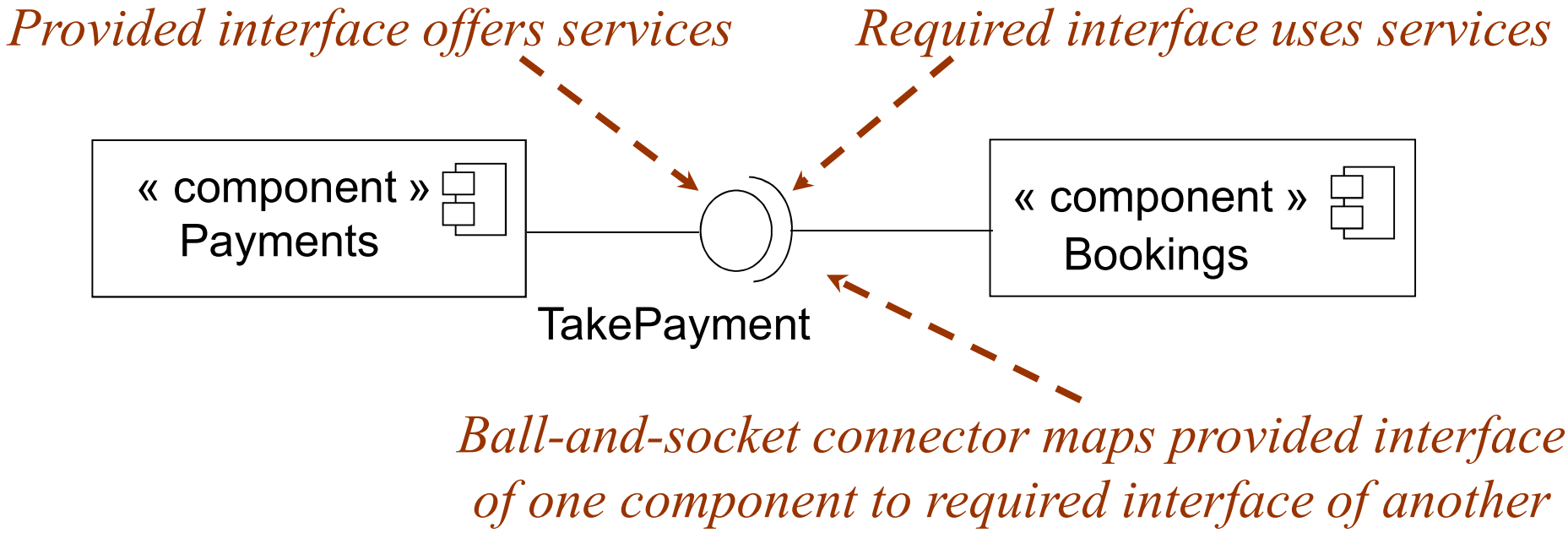
- This is (probably) composition:
  - Ingredient is in only one meal at a time
  - If you drop your dinner on the floor, you probably lose the ingredients too

Modelling Components in UML

- Standard UML techniques can be used to model components
- Component internals can be detailed in a class diagram
- Component interaction can be shown in a communication diagram

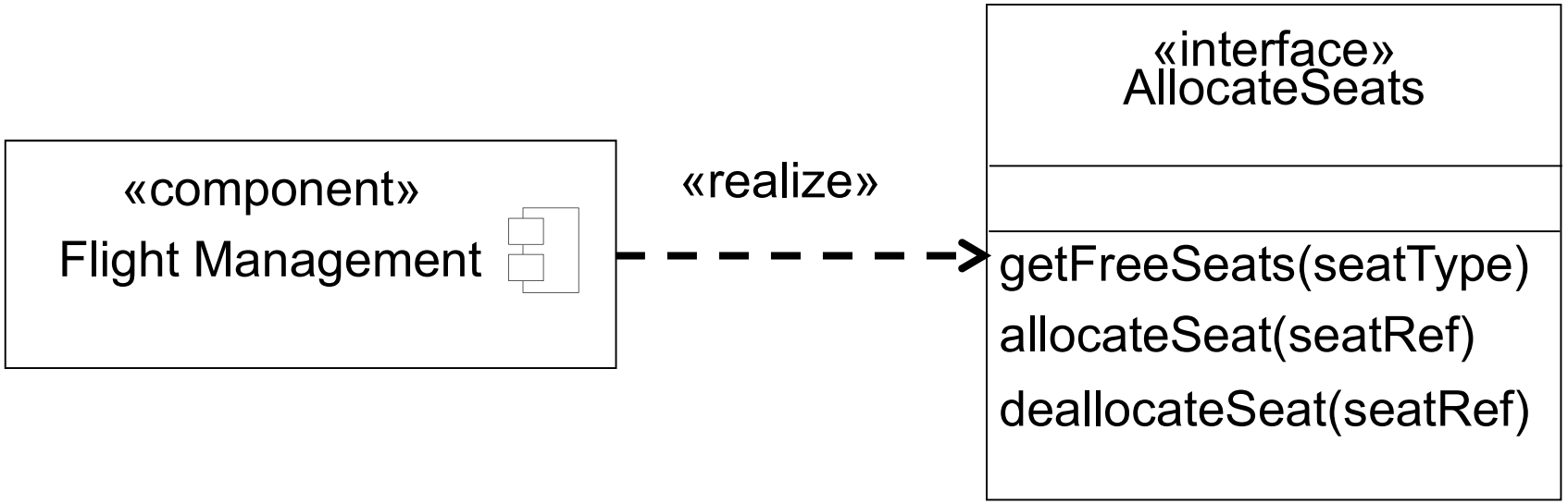
Modelling Components in UML

- UML has icons for modelling components in structure diagrams (e.g. class diagrams)



Modelling Components in UML

- Structure diagrams can mix component icons with other icons, e.g. interfaces



Software Development Patterns

A pattern:

- “describes a problem which occurs over and over again in our environment, and then describes the core of a solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.”

Alexander et al. (1977)

Software Development Patterns

- A pattern has:
  - A *context* = a set of circumstances or preconditions for the problem to occur
  - *Forces* = the issues that must be addressed
  - A software configuration that resolves the forces

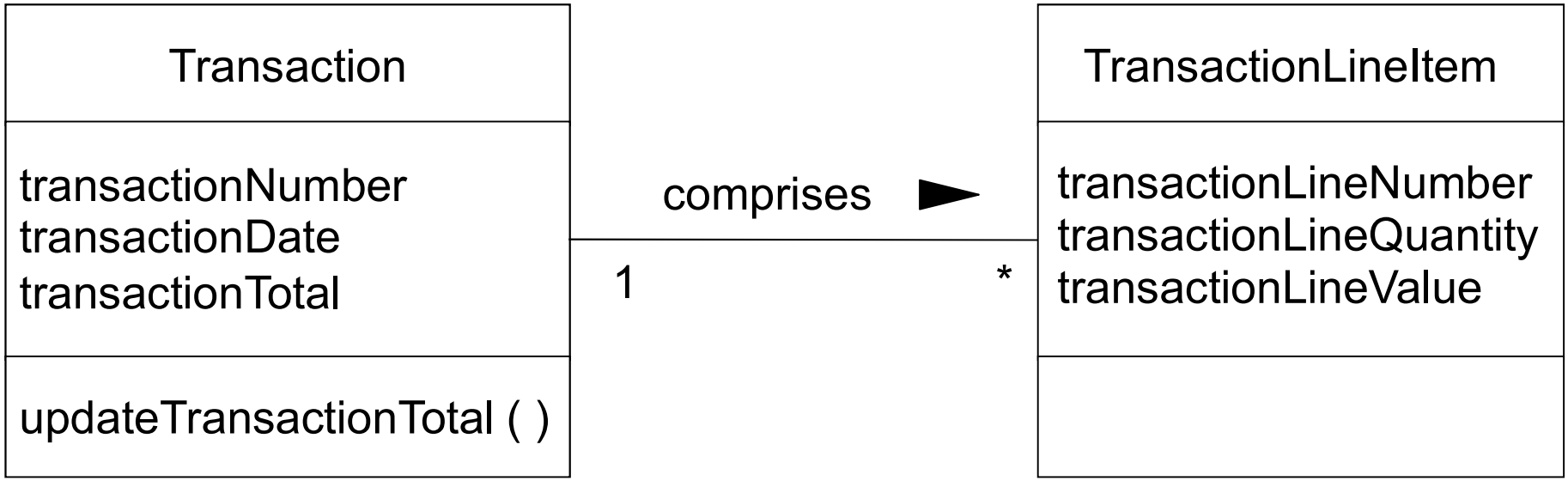
Software Development Patterns

- Patterns are found at many points in the systems development lifecycle:
  - Analysis patterns are groups of concepts useful in modelling requirements
  - Architectural patterns describe the structure of major components of a software system
  - Design patterns describe the structure and interaction of smaller software components

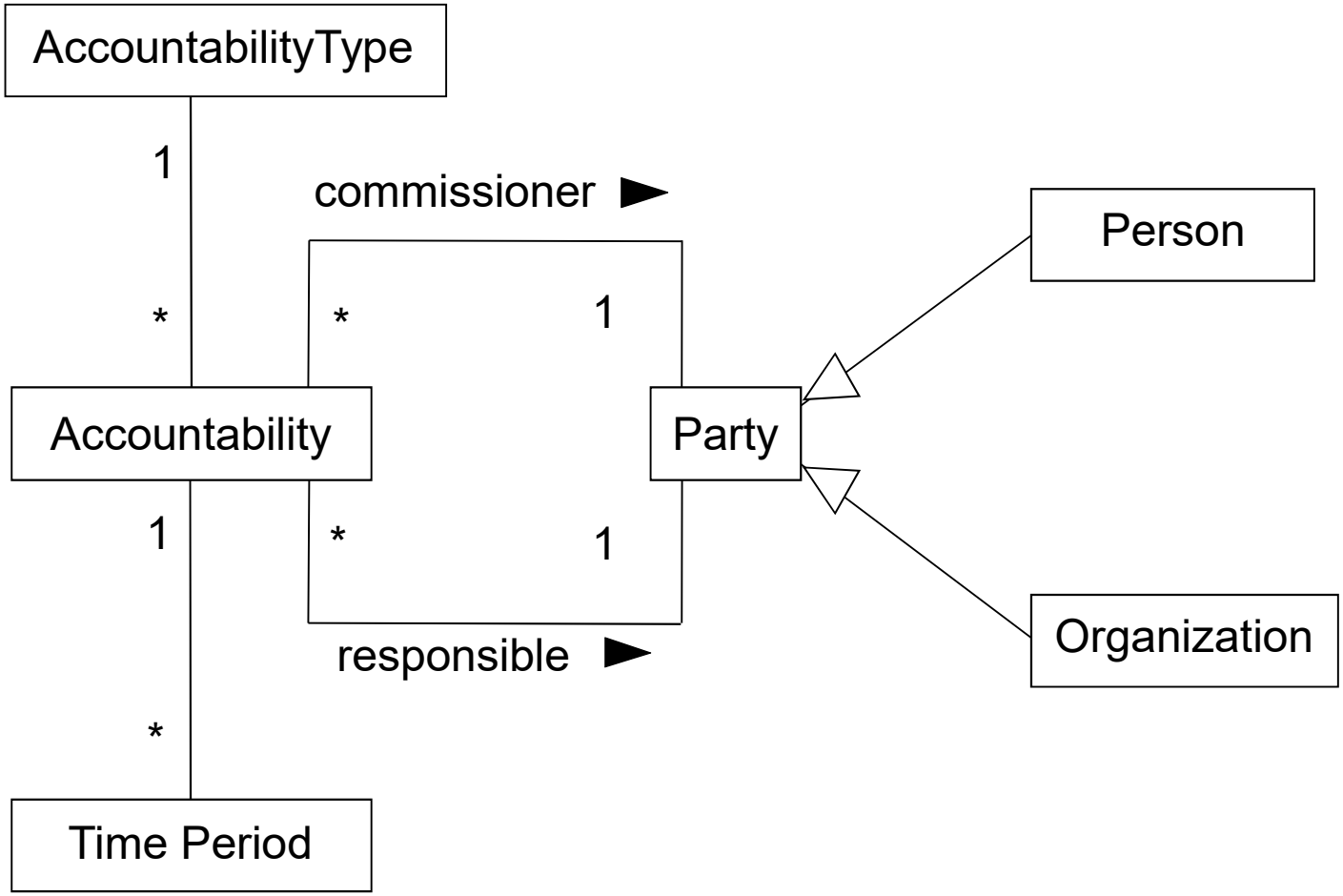
Software Development Patterns

- Patterns have been applied widely in software development:
  - Organisation patterns describe structures, roles and interactions in the software development organisation itself
- Antipatterns document bad practice
  - Mushroom Management is an organisation anti-pattern

Simplest Analysis Pattern



Accountability Analysis Pattern



Summary

In this lecture you have learned about:

- How to identify and model aggregation, composition and generalization
- Reusable components, and how to model them in structure diagrams
- What is meant by ‘pattern’, and how patterns are used in software development

References

- Bennett, McRobb and Farmer (2010)
- Cheesman and Daniels (2001)

  (For full bibliographic details, see Bennett, McRobb and Farmer)