E-course

What Is Object-Orientation?

E-course

REFINING THE REQUIREMENTS MODEL

In This Lecture You Will Learn:

E-course

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

Reuse in Software Development

E-course

- 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

E-course

- 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

E-course

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

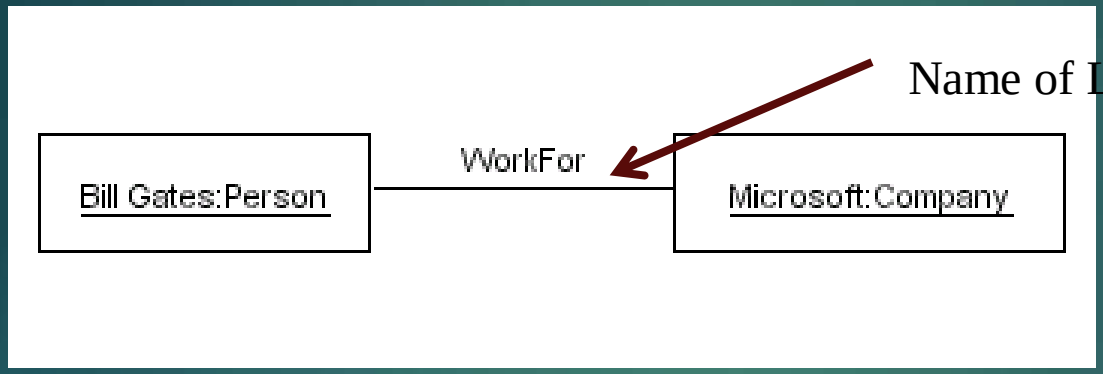
Association

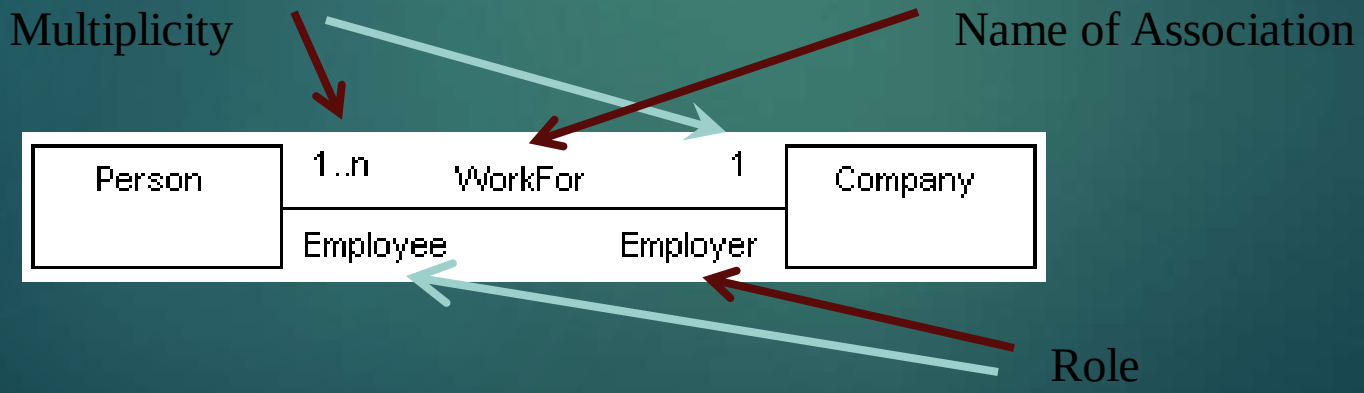
E-course

- Object-oriented systems are made up of objects of many classes. Associations represent relationships among classes.
- An association is represented by a line drawn between the associated classes involved with an optional role name attached to either end.
- The **role name** is used to specify the role of an associated class in the association.
- If an association connects two objects instead of classes, it is called a link.
- A link is an instance of an association.

Association (cont'd)

E-course





Qualification

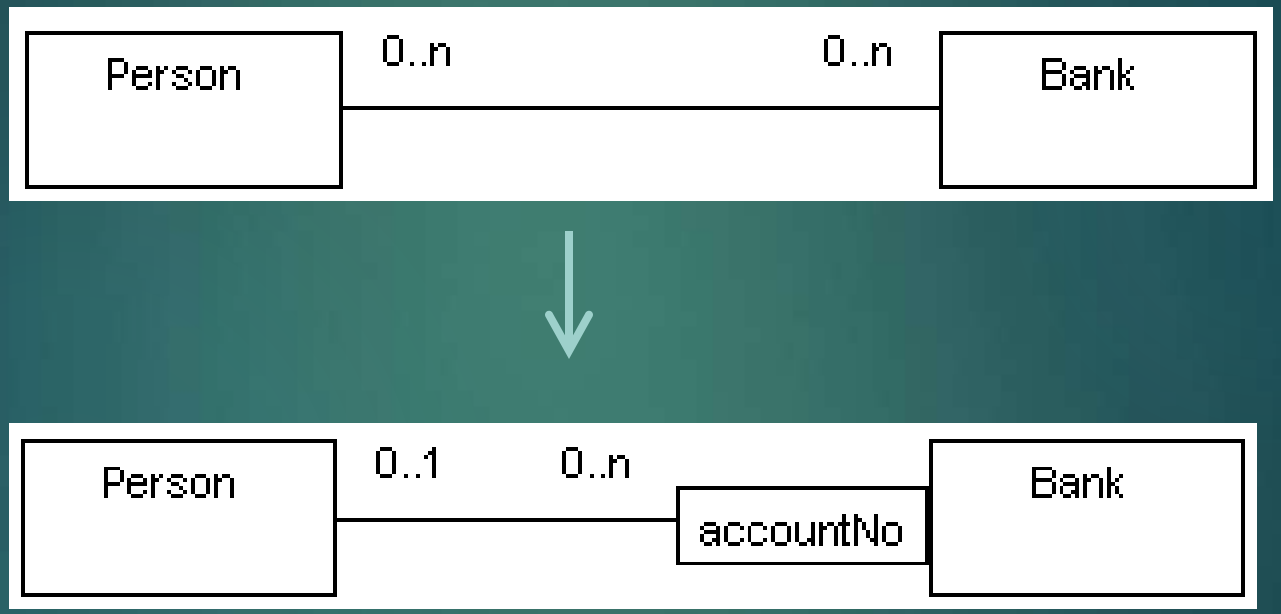
E-course

- *Qualification* serves as names or keys that are part of the association and are used to select objects across the association at the other end.

- In UML, a qualifier is used to model this association semantic, that is an association attribute whose value determines a unique object or a subset of objects at the other end of an association.

Qualification

E-course

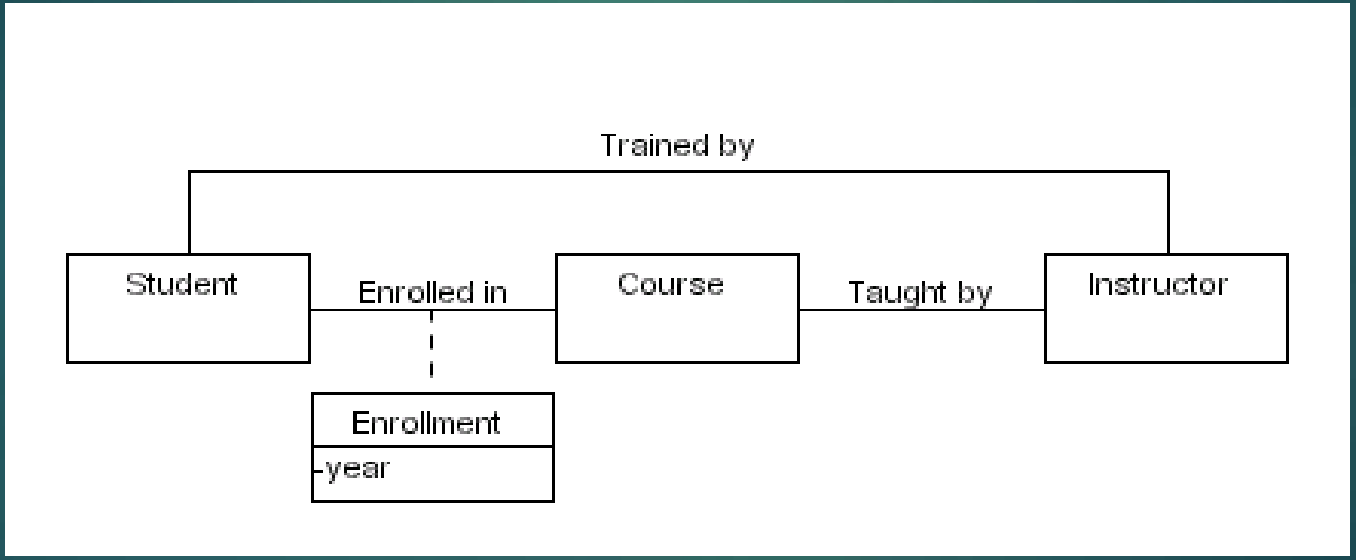


Bank + account no => person

Association Classes

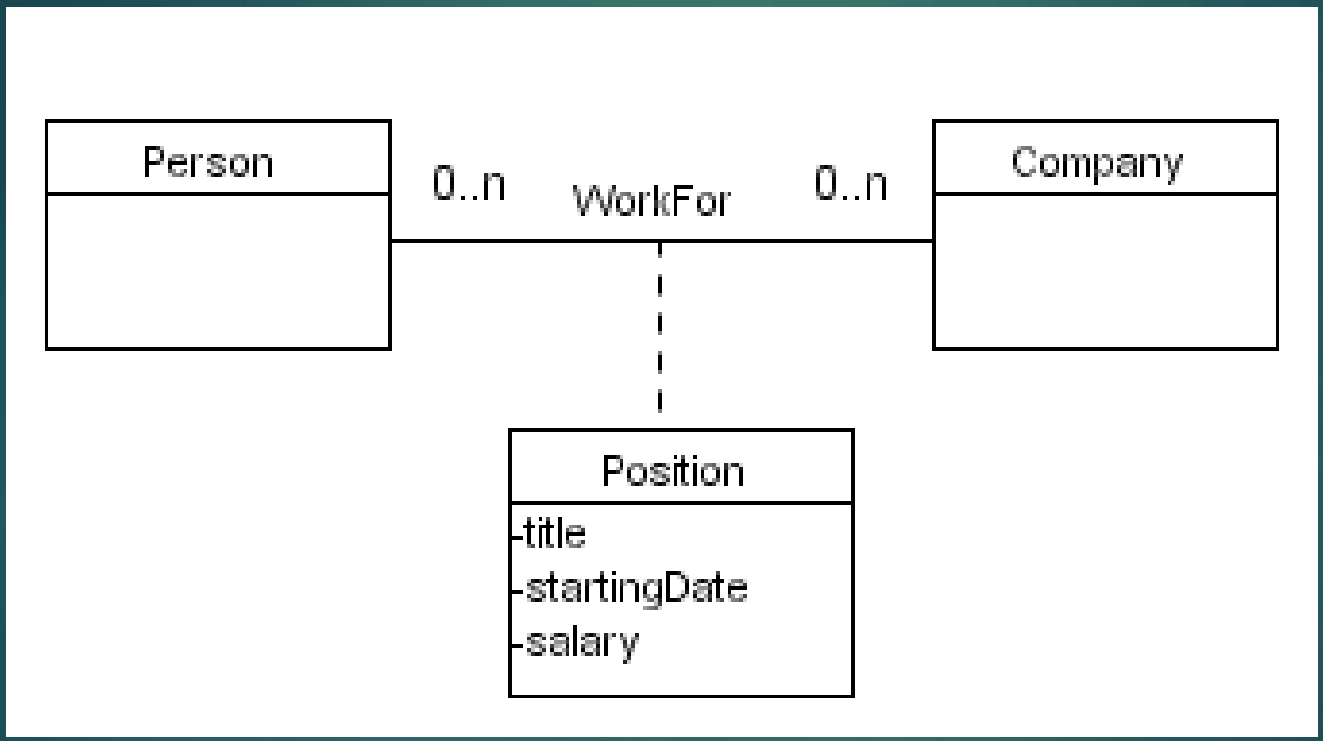
E-course

Sometimes, it is necessary to describe an association by including some attributes which do not naturally belong to the objects involved in the association.



Association Classes (cont'd)

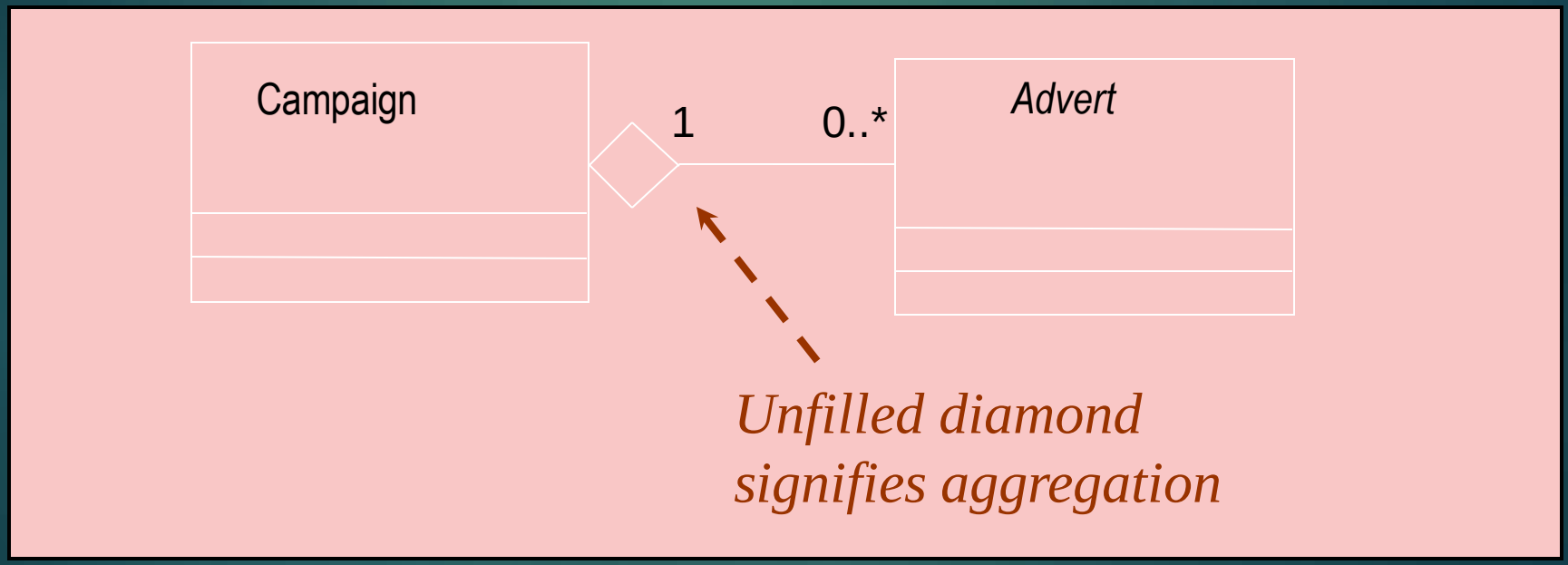
E-course



Aggregation and Composition

E-course

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



Aggregation and Composition

E-course

- 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

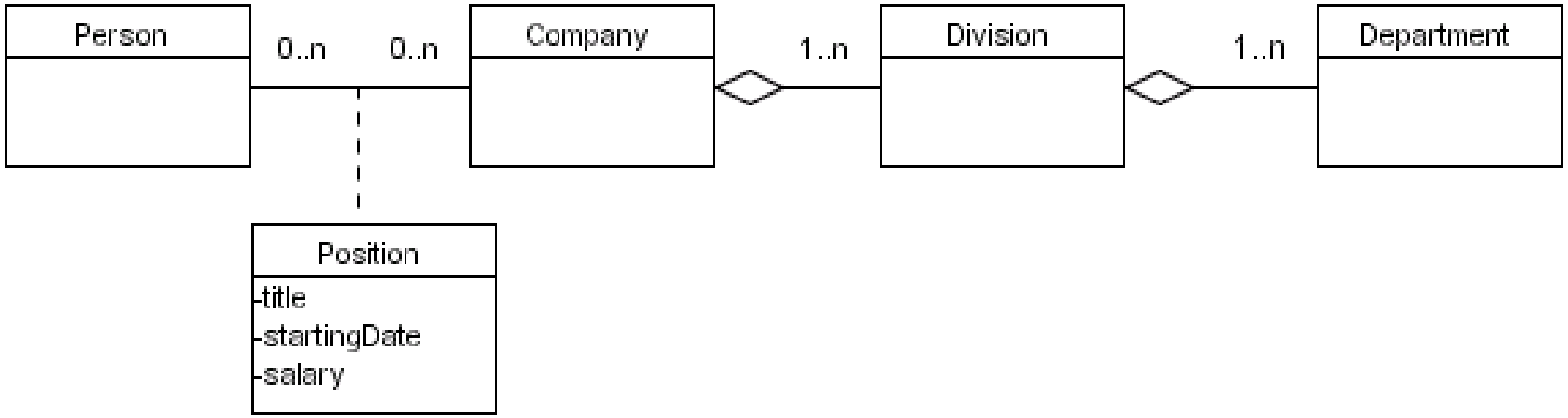
Aggregation

E-course

- *Aggregation* is a stronger form of association.
- It represents the has-a or part-of relationship.
- In UML, a link is placed between the “whole” and the “parts” classes with a diamond head attached to the “whole” class to indicate that this association is an aggregation.
- Multiplicity can be specified at the end of the association for each of the “part-of” classes to indicate the quantity of the constituent parts.
- Typically, aggregations are not named, and the keywords used to identify aggregations are “consists of”, “contains” or “is part of”.

Aggregation (cont'd)

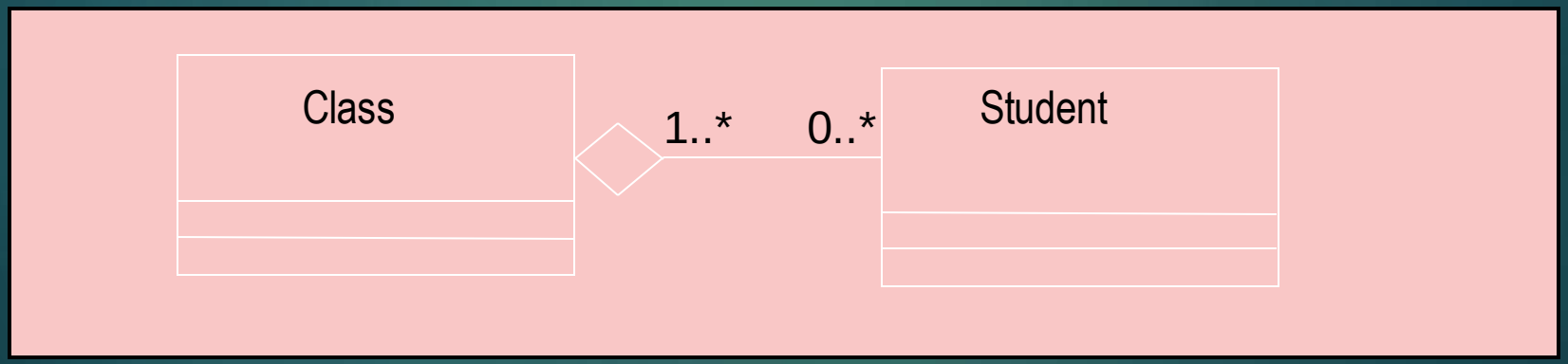
E-course



Aggregation and Composition

E-course

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



Composition

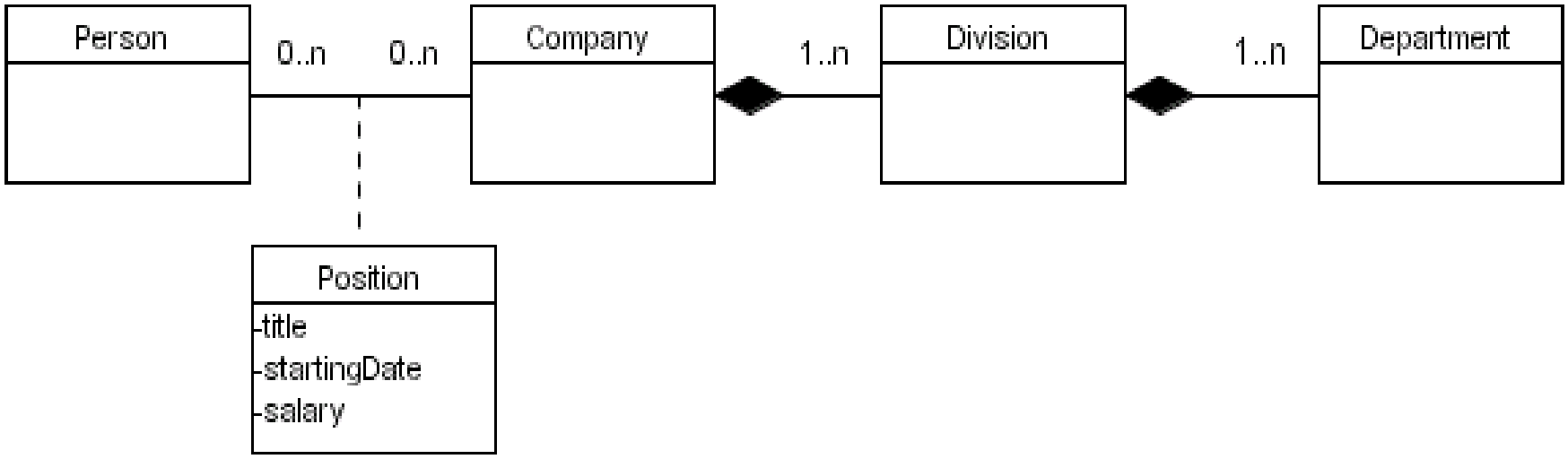
E-course

- A stronger form of aggregation is called *composition*, which implies exclusive ownership of the “part-of” classes by the “whole” class, i.e. a composite object has exclusive ownership of the parts objects.

- This means that parts may be created after a composite is created, but such parts will be explicitly removed before the destruction of the composite.

Composition (cont'd)

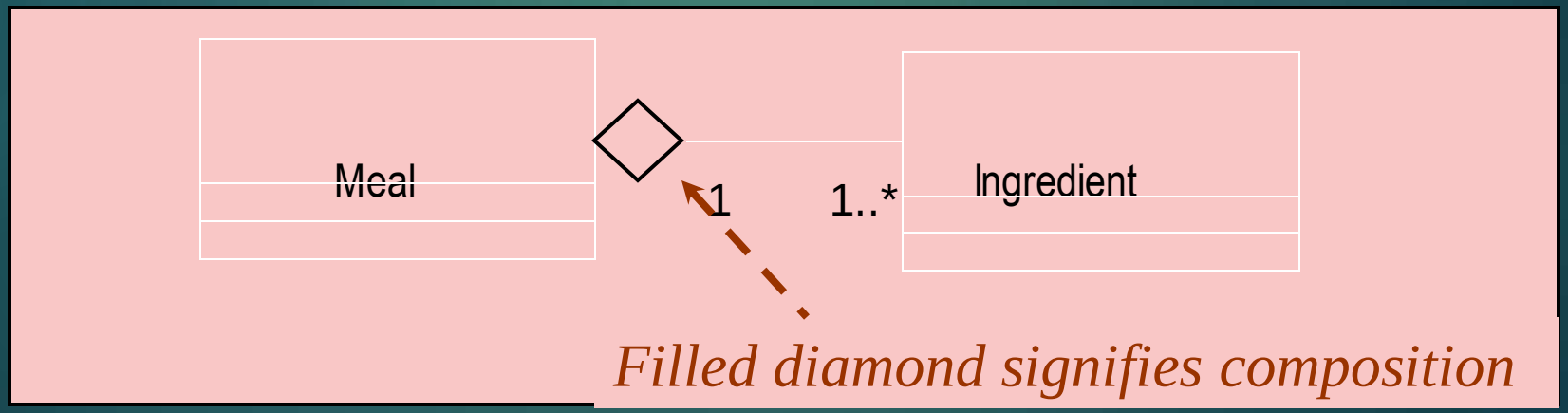
E-course



Aggregation and Composition

E-course

- Another everyday example:
- 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



Constraints and Notes

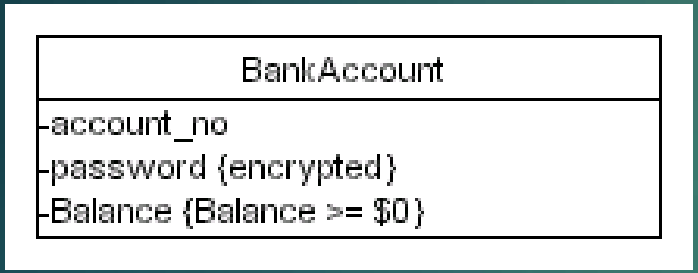
E-course

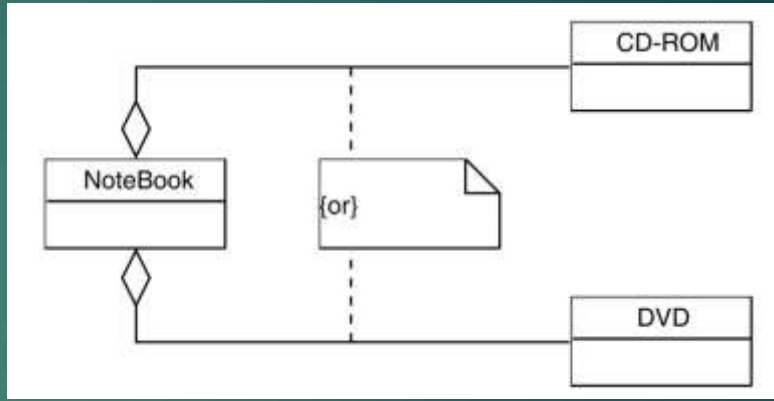
- *Constraints* are an extension of the semantics of a UML element, allowing one to add new rules or modify existing ones. Sometimes it is helpful to present an idea about restrictions on attributes and associations for which there is no specific notation.

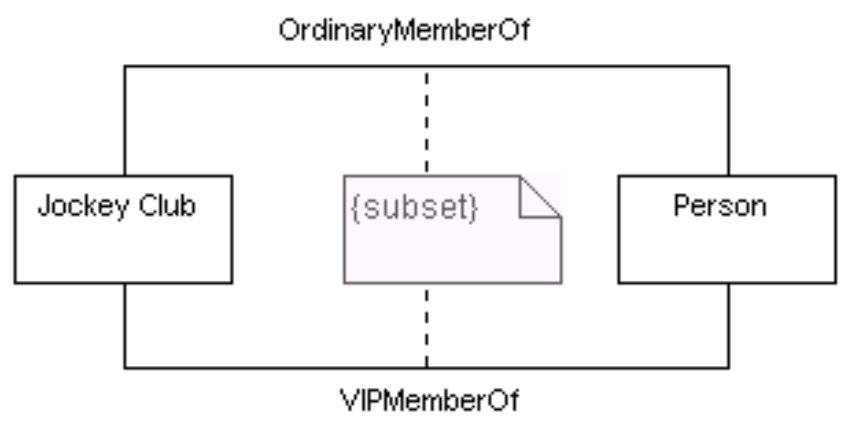
- Constraints are represented by a label in curly brackets ({constraintName} or {expression}) that are attached to the constrained element.

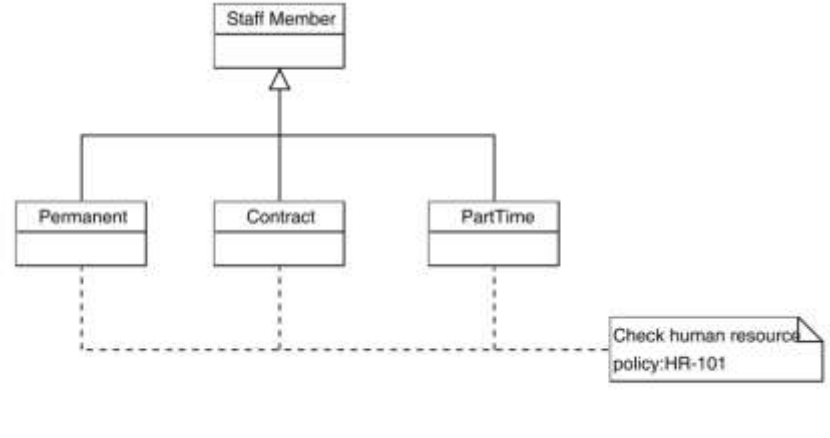
Constraints and Notes (cont'd)

E-course



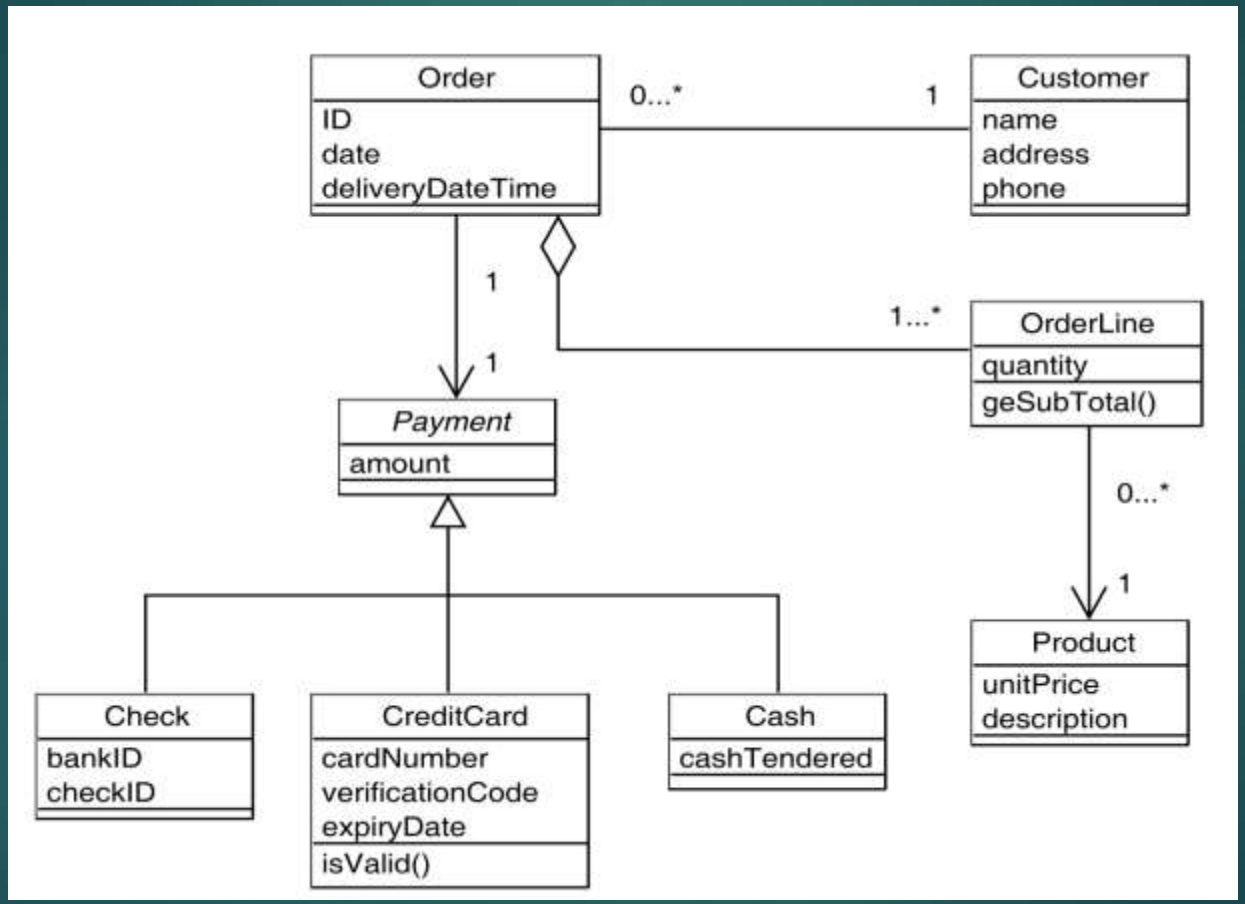






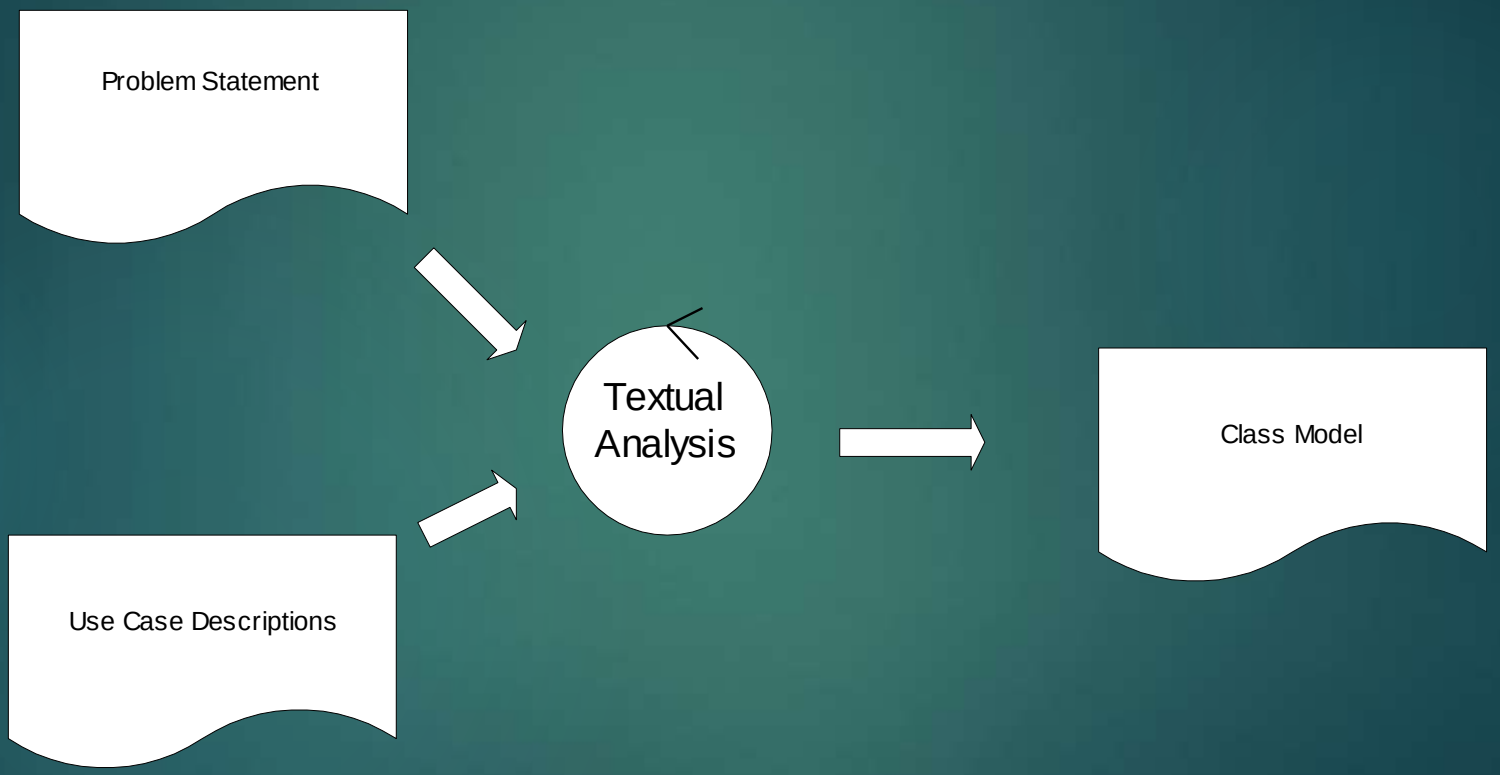
Example – A Sales Order System

E-course



Structural Analysis Techniques

E-course



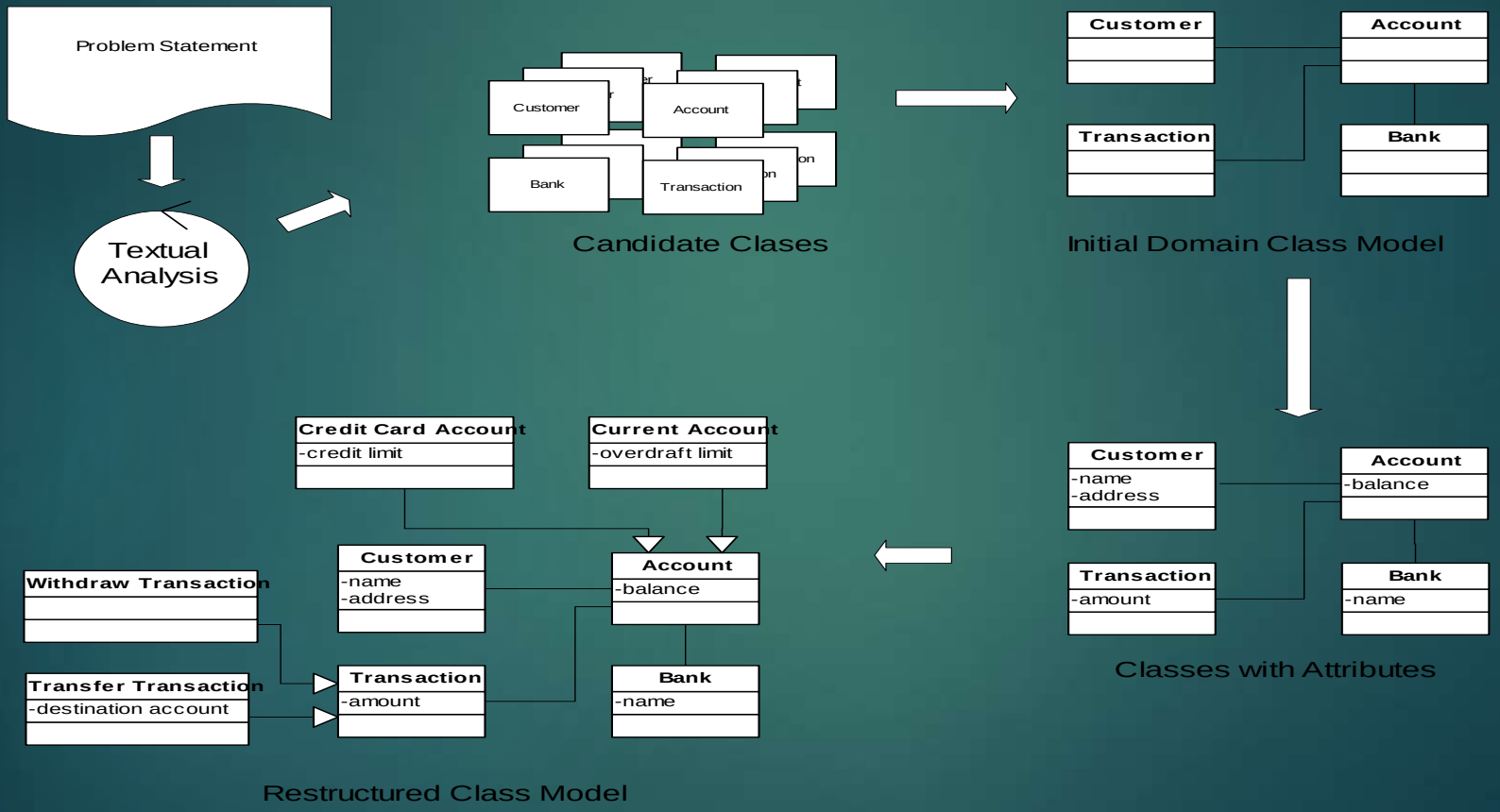
Domain Modeling and Analysis

E-course

- Preparing the problem statement;
- Identifying objects and classes using textual analysis;
- Developing a data dictionary;
- Identifying associations between classes;
- Identifying attributes of classes and association classes;
- Structuring classes using inheritance;
- Verifying access paths for likely queries; and
- Iterating and refining the model.

Domain Modeling and Analysis (cont'd)

E-course



Identifying objects and classes

E-course

- To identify objects and classes, perform a textual analysis to extract all nouns and noun phrases from the problem statement.
- Nouns or noun phrases of the following categories are more likely to represent objects:
  - Tangible things (e.g. classroom, playground)
  - Conceptual things (e.g. course, module)
  - Events (e.g. test, examination, seminar)
  - Outside organizations (e.g. publisher, supplier)
  - Roles played (e.g. student, teacher, principal)
  - Other systems (e.g. admission system, grade reporting system)

Adding Generalisation Structure

E-course

- Add generalisation 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

Identifying Associations between Classes

E-course

- An association is a relationship between objects.
- For example, John and Peter are instances of the class Person and John is the father of Peter.
- Association can be identified by looking for verbs and verb phrases connecting two or more objects in the problem statement.
- e.g. “A client may **open** one or more accounts for stock trading.”
  - Name of the association: **has** or **opened by**?
- The name of the association should reflect the nature rather than the historic event.

Identifying Associations between Classes (cont'd)

E-course

Verb phrase	Association
A client may open one or more accounts for stock trading.	has
When a client issues a buy order for an account, the client must specify the stock code, number of shares and the maximum price that he is willing to pay for them (the bid price).	issued by, buy
When a client issues a sell order for an account, the client must specify the stock code, the number of share and the minimum price that he is willing to sell them (the ask price).	issued by, sell
All trade orders will be forwarded to the stock trading system of the stock exchange for execution.	executed by
When an order is completed, the stock trading system of the stock exchange will return the transaction details of the order to the online stock trading system.	returned by
The transaction details of a trade order may be a list of transactions, and each transaction specifies the price and the number of shares traded.	consists of

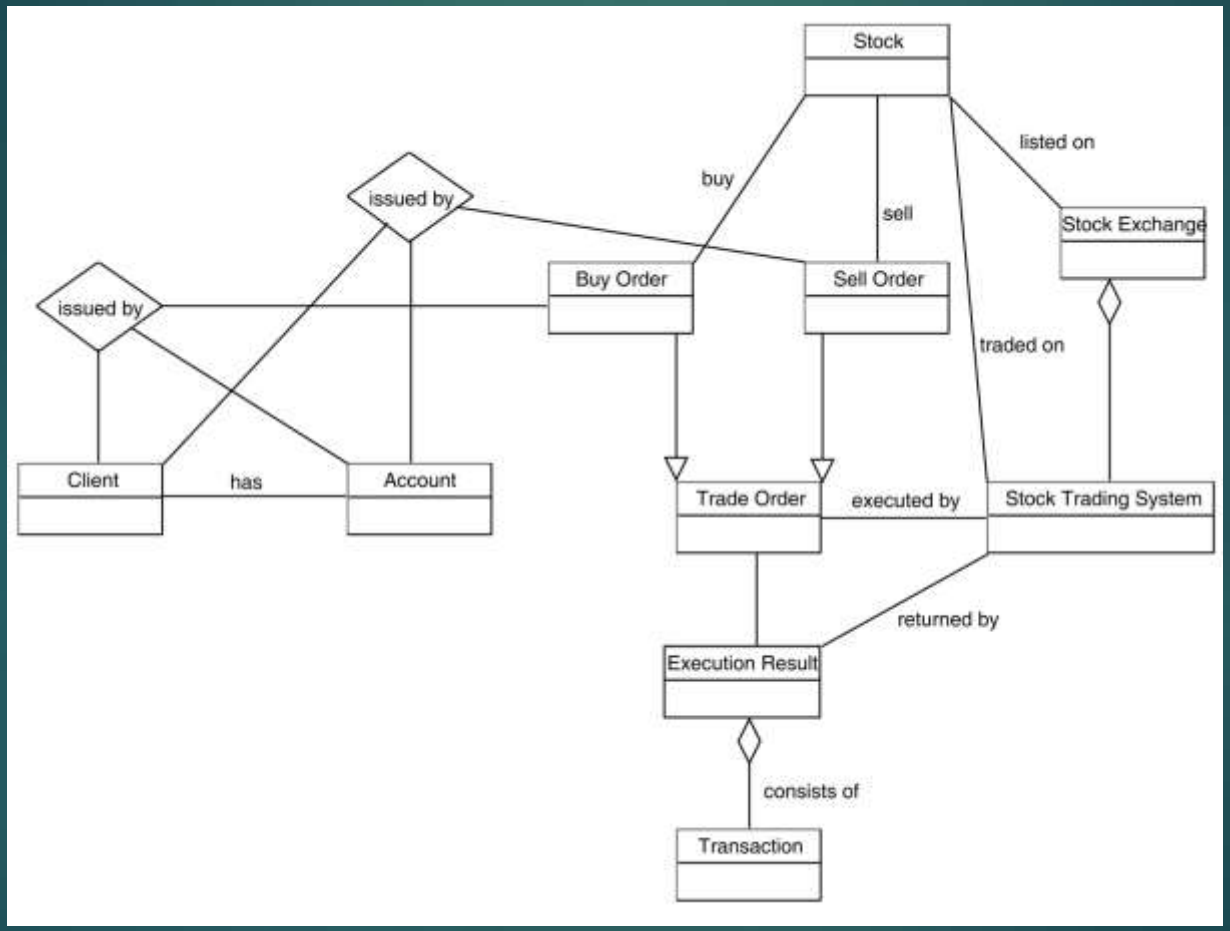
Identifying Associations between Classes (cont'd)

E-course

- From the domain knowledge, we have the following associations:
  - A stock is listed on a stock exchange.
  - A stock is traded on a stock trading system of a stock exchange.
  - The result of a trade order is a list of transactions.
  - A stock exchange has one or more stock trading systems.

Initial Domain Class Diagram

E-course



Identifying Attributes of Classes and Association Classes

E-course

- Attributes are properties of a class, such as name, address, telephone number of the Client class.
- Look for nouns or noun phrases followed by possessive phrases, e.g. “address of the client”.
- Adjectives that appear immediately before a noun corresponding to a class can also be an enumerated value of an attribute, e.g. “a canceled buy order”.
- Attributes are less likely to be discovered from the problem statement.
- However, it is not necessary to identify all attributes in this step because the attributes do not affect the structure of the domain class model.
- Instead we should only do so if they can be identified readily. We will be able to identify the attributes more readily at later stages of the development life cycle (e.g. detailed design phase).

Adding Structure:

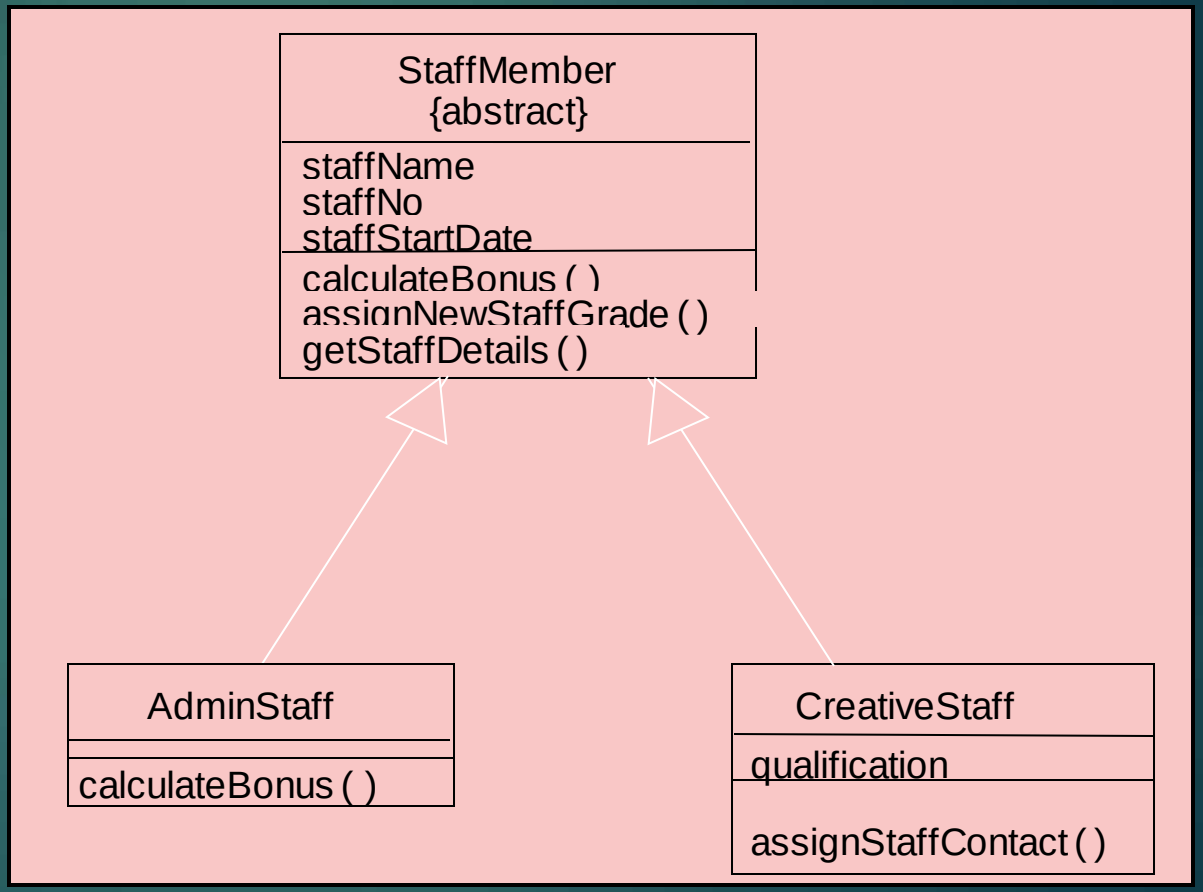
E-course

- 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:

E-course



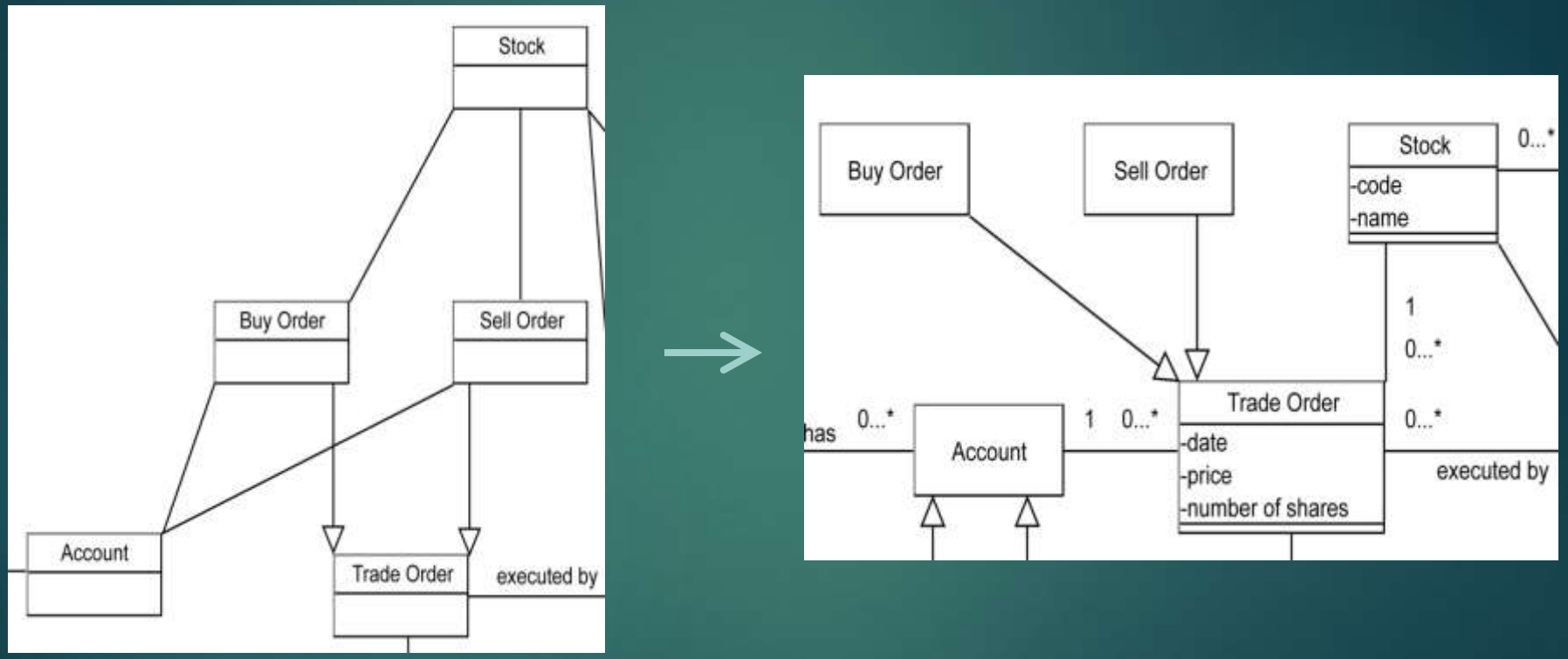
Structuring Classes Using Inheritance

E-course

- Two approaches: top down and bottom up
- Bottom-up approach:
  - similar class name
  - similar attributes
  - similar operations
  - similar associations
- e.g. the buy order and sell order classes both have the price and number of shares attributes and both of them are associated with the stock class and account class.

Structuring Classes Using Inheritance (cont'd)

E-course



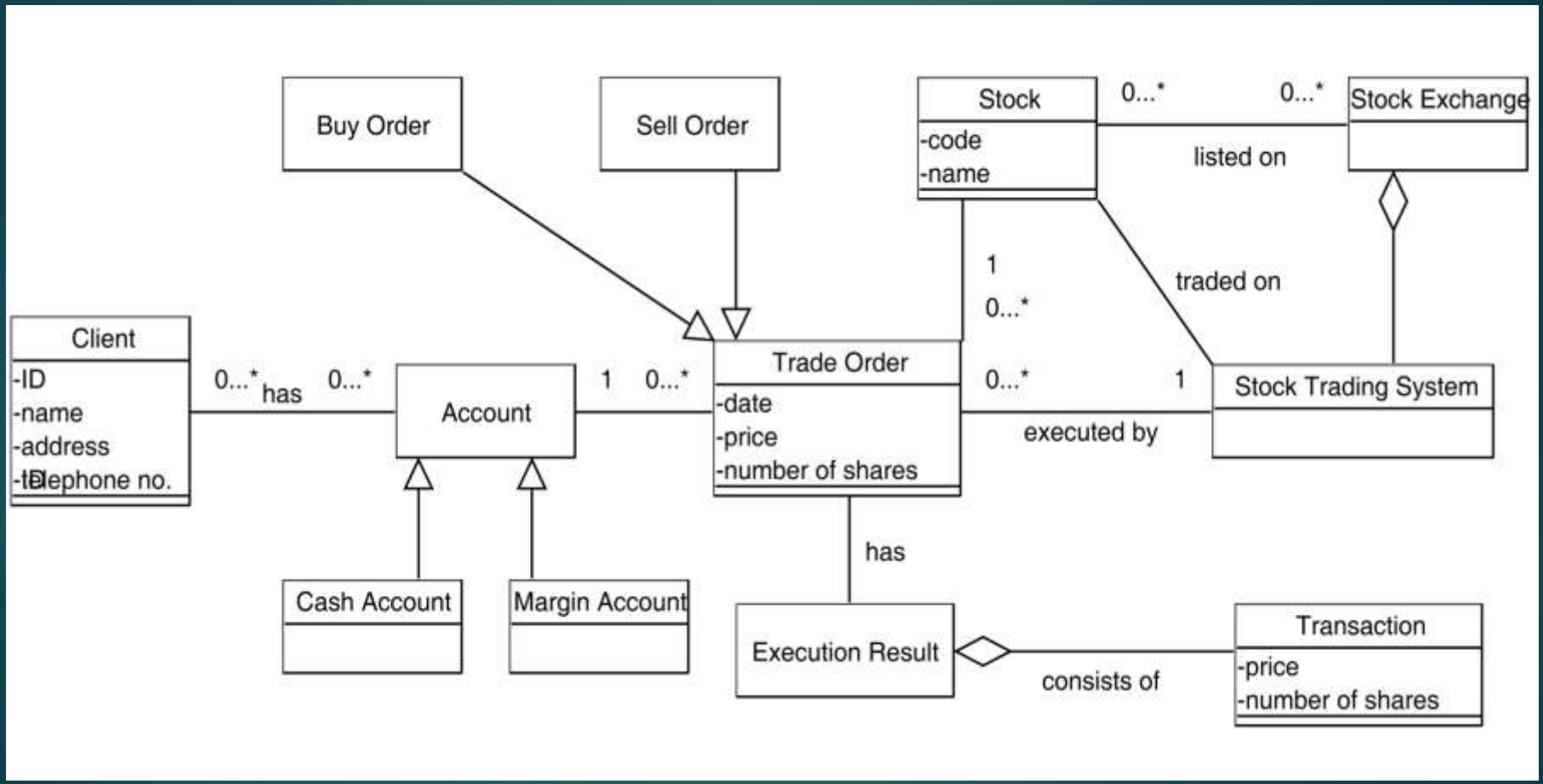
Structuring Classes Using Inheritance (cont'd)

E-course

- Top-down approach:
  - check a class to see whether it has some special cases that have additional structural or behavioral requirements.
  - noun phrases consisting of adjectives and class names,
    - e.g. sell order class and buy order class
  - taxonomies of real-life objects
  - domain knowledge,
    - e.g. cash account and margin account

Structuring Classes Using Inheritance (cont'd)

E-course



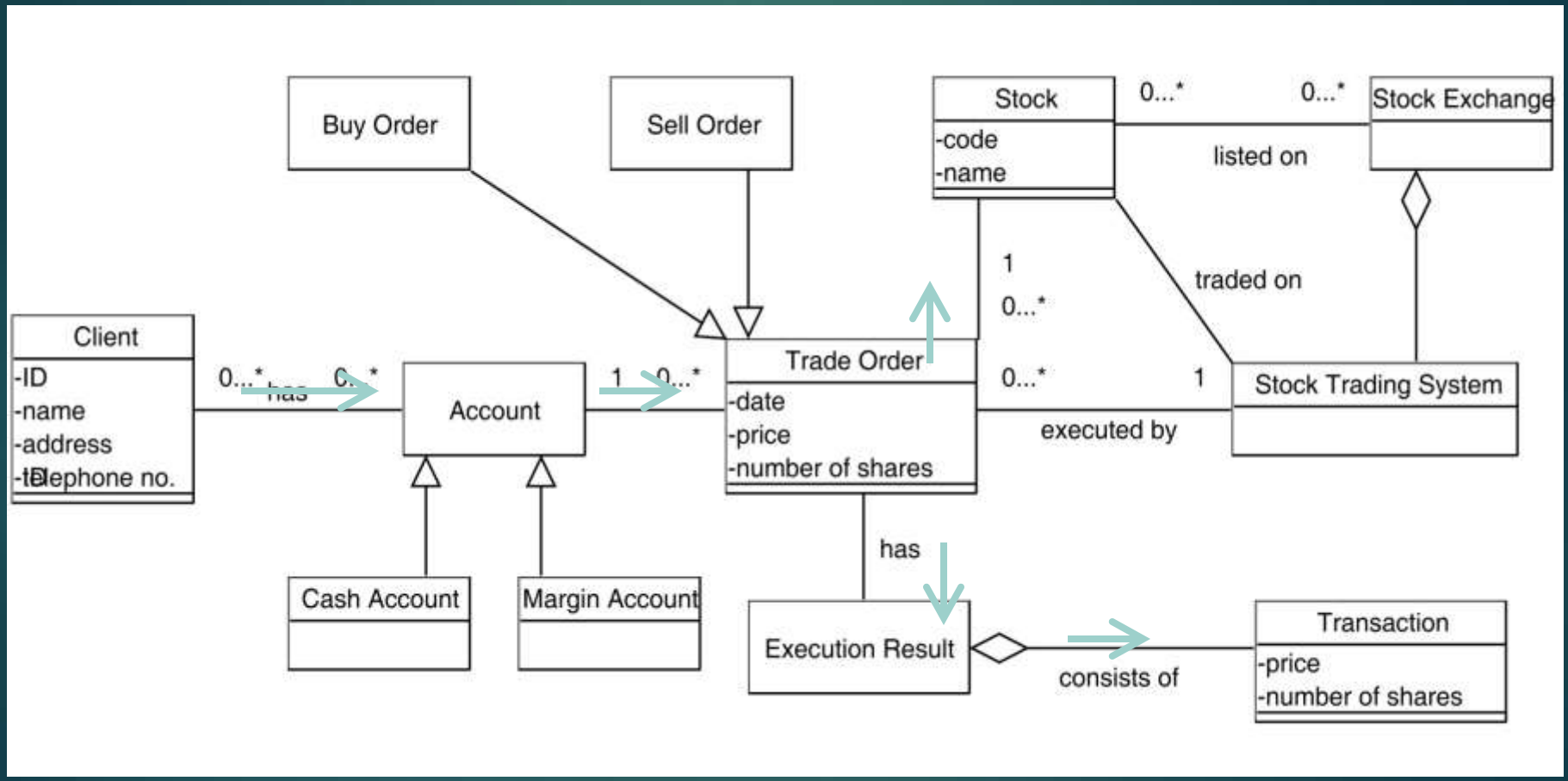
Verifying Access Paths for Likely Queries (cont'd)

E-course

- Check whether the domain class diagram can provide the correct answers to queries that are common to other applications in the domain.
- In the online stock trading system example, a typical client query would be the current stock balance of his account.
- This requires an association between the account class and the stock class to provide the information on the number of shares held in the account.

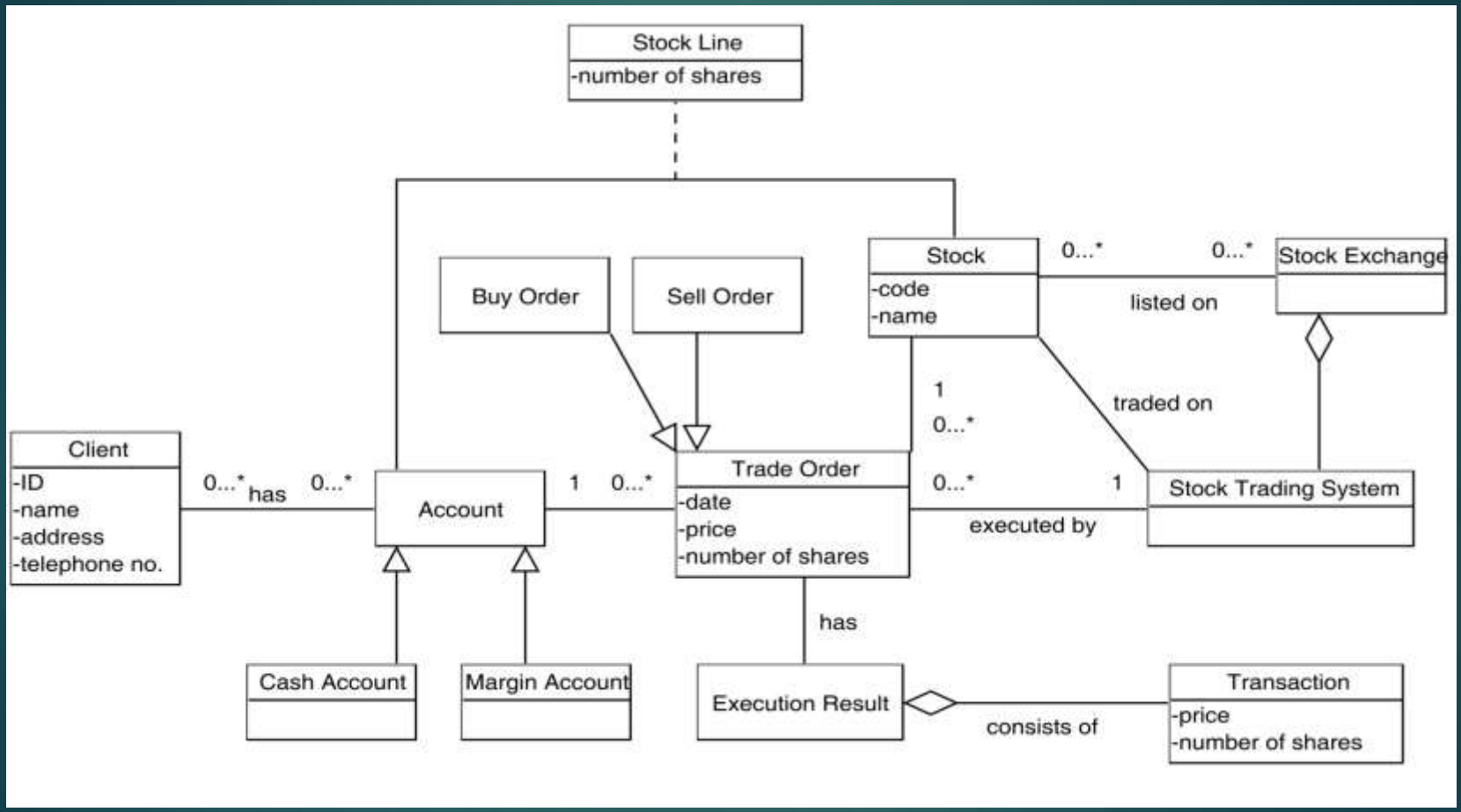
Verifying Access Paths for Likely Queries (cont'd)

E-course



Verifying Access Paths for Likely Queries (cont'd)

E-course



Iterating and Refining the Model

E-course

- It is highly unlikely that we are able to develop the correct domain class model in one pass.
- The domain class model needs to be refined several times before it becomes robust.
- The development of the domain class model is not a rigid process, and we need to repeatedly apply the above steps until the domain class model finally becomes stable.

Iterating and Refining the Model (cont'd)

E-course

- The following checklist can help you identify areas to improve the domain class model .
  - A class without attributes, operations and associations - consider removing the class.
  - A class with many attributes and operations covering a wide area of requirements - consider splitting the class into two or more classes.
  - A query cannot be answered by tracing the domain class model - consider adding additional associations.
  - Asymmetries in generalizations and associations - consider adding additional associations and restructuring the classes with inheritance.
  - Attributes or operations without a hosting class - consider adding new classes to hold these attributes and operations.

Modelling Components in UML

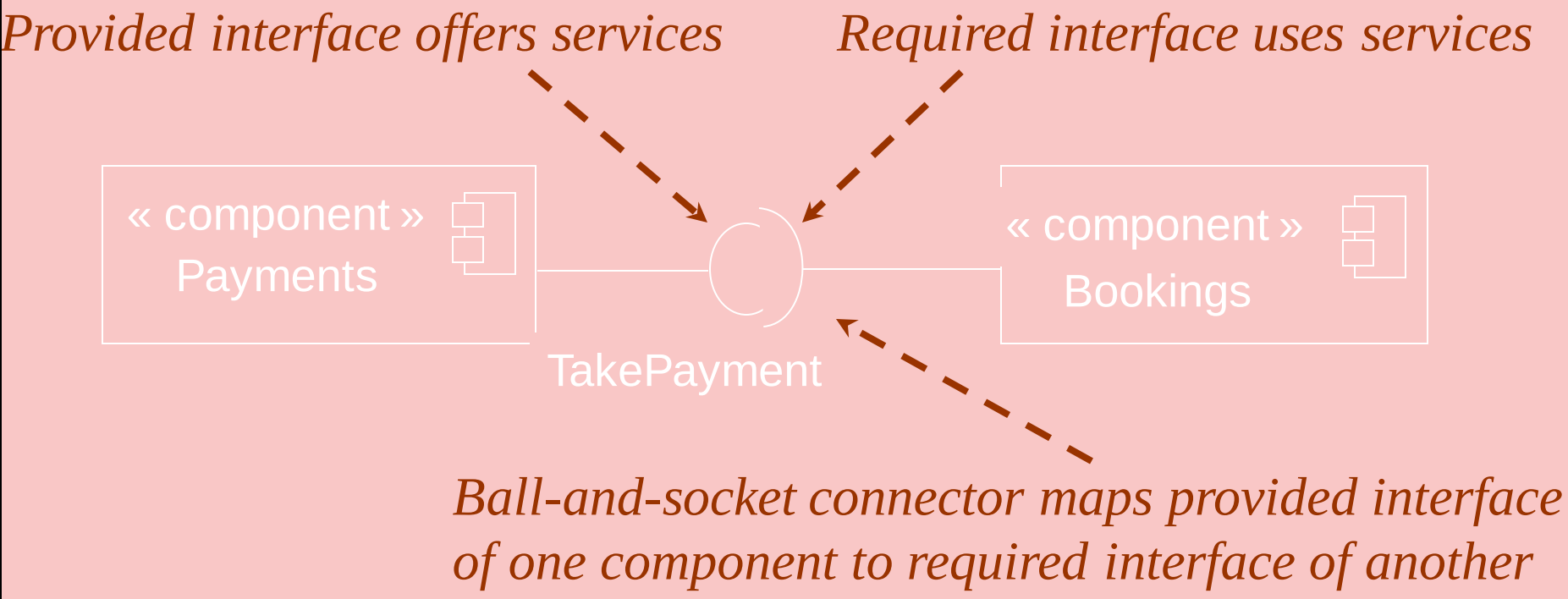
E-course

- 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

E-course

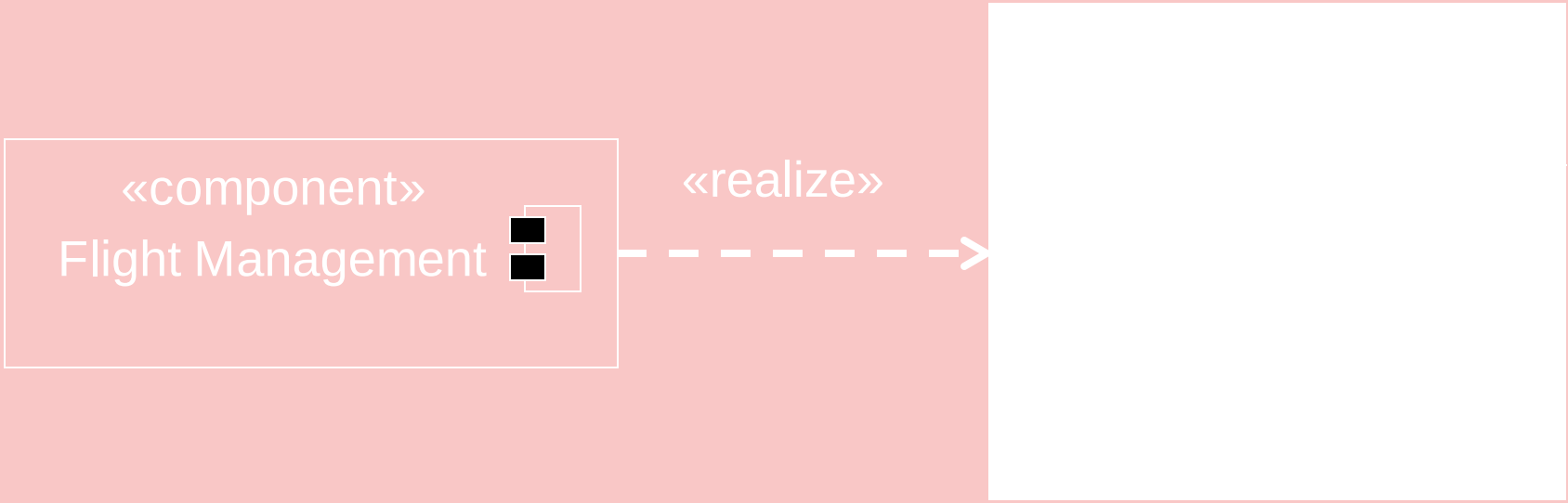
- UML has icons for modelling components in structure diagrams (e.g. class diagrams)



Modelling Components in UML

E-course

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



Software Development Patterns

E-course

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

E-course

- 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

E-course

- 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

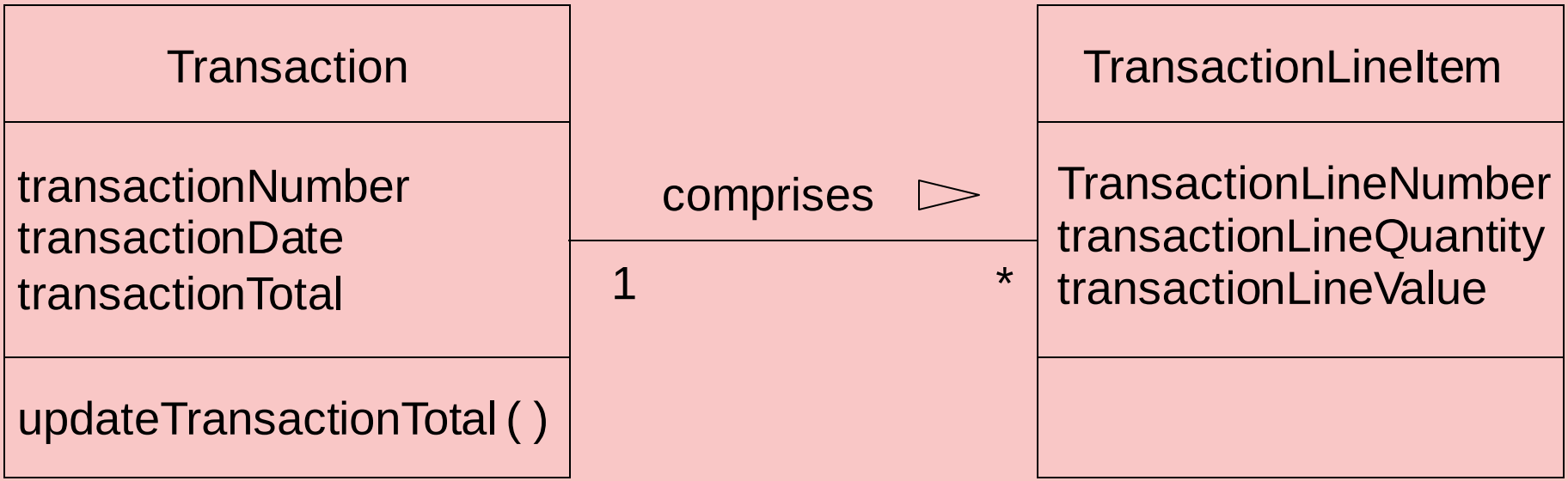
Software Development Patterns

E-course

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

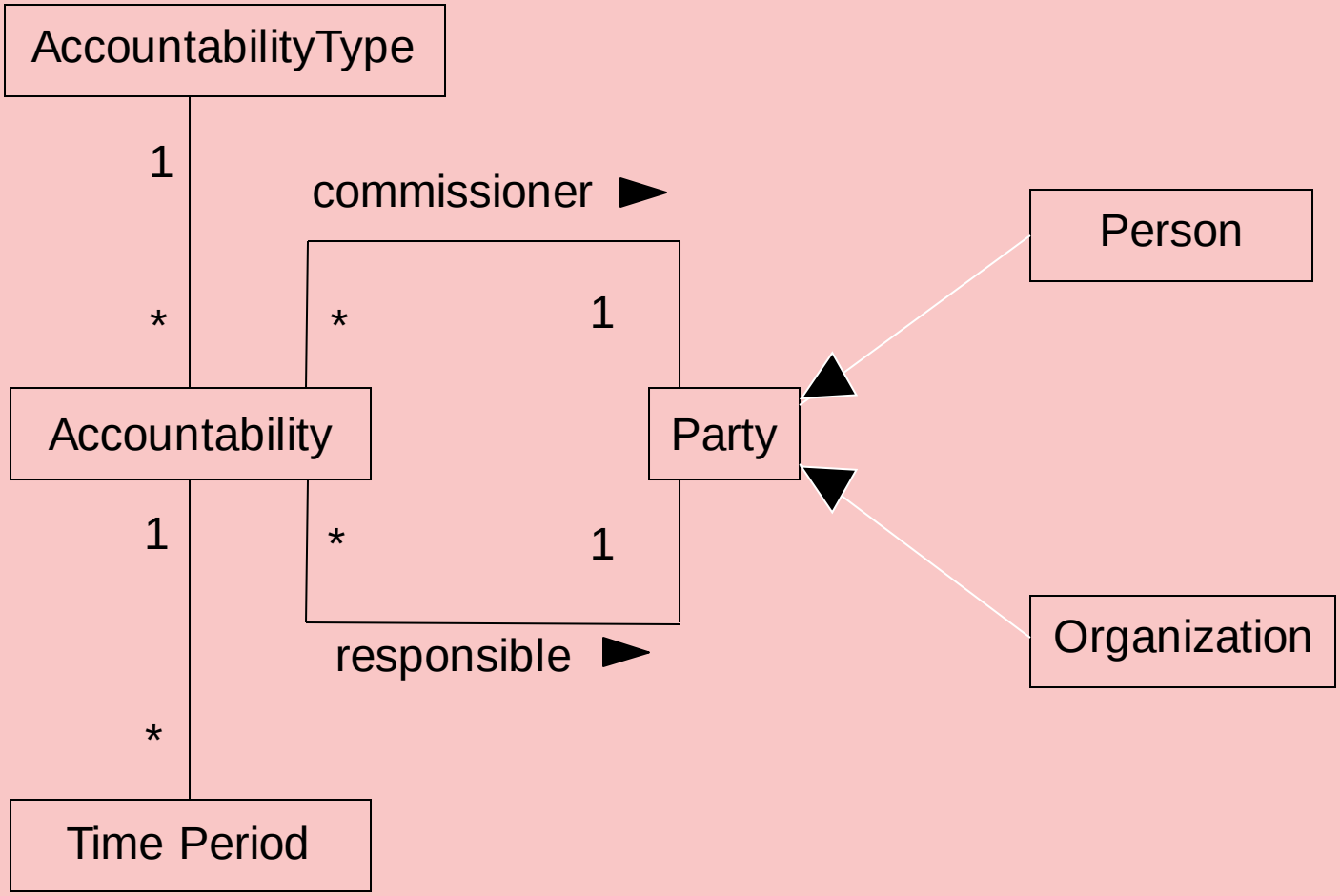
Simple Analysis Pattern

E-course



Accountability Analysis Pattern

E-course



Analysis Patterns

E-course

Pattern name: A descriptor that captures the essence of the pattern.

Intent: Describes what the pattern accomplishes or represents

Motivation: A scenario that illustrates how the pattern can be used to address the problem.

Forces and context: A description of external issues (forces) that can affect how the pattern is used and also the external issues that will be resolved when the pattern is applied.

Solution: A description of how the pattern is applied to solve the problem with an emphasis on structural and behavioral issues.

Consequences: Addresses what happens when the pattern is applied and what trade-offs exist during its application.

Design: Discusses how the analysis pattern can be achieved through the use of known design patterns.

Known uses: Examples of uses within actual systems.

Related patterns: On e or more analysis patterns that are related to the named pattern because (1) it is commonly used with the named pattern; (2) it is structurally similar to the named pattern; (3) it is a variation of the named pattern.

References

E-course

- Bennett, McRobb and Farmer (2005)
- Ambler (2003)
- Cheesman and Daniels (2001)
- Coad (1997)
- Fowler (1997)

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

- Object-Oriented Technology - From Diagram to Code with Visual Paradigm for UML, Curtis H.K. Tsang, Clarence S.W. Lau and Y.K. Leung, McGraw-Hill Education (Asia), 2005

WE FOCUS ON KNOWLEDGE-BASED ON EDUCATION

KSRA of Empowerment is a global non-profit organization committed to bringing empowerment through education by utilizing innovative mobile technology and educational research from experts and scientists. KSRA emerged in 2012 as a catalytic force to reach the hard to reach populations worldwide through Learning management system & E-learning & mobile learning.

The KSRA team partners with local under-served communities around the world to improve the access to and quality of knowledge based on education, amplify and augment learning programs where they exist, and create new opportunities for e-learning where traditional education systems are lacking or non-existent.

E-course