

WIA2002: Software Modelling

Semester 1, Session 2016/17

Lecture 5: Structural Modelling – UML Class
Diagrams

(Part 1: Analysis/Entity Class Diagrams)

Learning Objectives

- Understand how an analysis model differs from requirements and design models
- Understand the concepts of structural modelling
- Understand the basic elements of structural models
- Understand the techniques used for object identification
- Understand the Concepts and notation of the class diagrams:
 - Class and Object
 - Links, Associations and Multiplicities
 - Attributes, Operations and State

Why Analyse Requirements?

- Requirements (Use Case) model alone is not enough
 - There may be repetition
 - Some parts may already exist as standard components
 - Use cases give little information about structure of software system

Structural Modelling

- Object-oriented modelling allows the analyst to reduce the semantic gap between the underlying problem domain and the evolving structural model.
- A structural model is a formal way of representing the objects that are used and created by a software system.
- The goal of the analyst is to discover the key objects contained in the problem domain and to build a structural model.
- The structural model is drawn using an iterative process in which the model becomes more detailed and less conceptual over time.

Structural Modelling

- One of the primary purposes of the structural model is to create a vocabulary that can be used by the analyst and the users.
- Structural models represent the things, ideas, or concepts contained in the domain of the problem.
 - Also allow the representation of the relationships among the things, ideas, or concepts.
- By creating a structural model of the problem domain, the analyst creates the vocabulary necessary for the analyst and users to communicate effectively.

Structural Modelling

- At this stage of development, the structural model does not represent software components or classes in an object-oriented programming language.
- The refinement of these initial classes into programming-level objects comes later.
- The structural model at this point should represent the responsibilities of each class and the collaborations among the classes.
- Structural models are depicted using **CRC cards**, **class diagrams**, and, in some cases, **object diagrams**.

BASIC ELEMENTS OF STRUCTURAL MODELS

Objects (Recap)

An object is:

- “an abstraction of something in a problem domain, reflecting the capabilities of the system to
 - keep information about it,
 - interact with it,
 - or both.”

Coad and Yourdon (1990)

Objects (Recap)

“Objects have state, behaviour and identity.”

Booch (1994)

- *State*: the condition of an object at any moment, affecting how it can behave.
- *Behaviour*: what an object can do, how it can respond to events and stimuli.
- *Identity*: each object is unique.

Examples of Objects (Recap)

Object	Identity	Behaviour	State
A person.	'Hussain Pervez.'	Speak, walk, read.	Studying, resting, qualified.
A shirt.	My favourite button white denim shirt.	Shrink, stain, rip.	Pressed, dirty, worn.
A sale.	Sale no #0015, 18/05/05.	Earn loyalty points.	Invoiced, cancelled.
A bottle of ketchup.	<i>This</i> bottle of ketchup.	Spill in transit.	Unsold, opened, empty.

Types of object

- **Physical (tangible) objects** - those visible and touchable things (e.g. a book, a bus, a computer or a Christmas tree).
- **Conceptual objects** - intangible things, (e.g. a bank account, a time schedule).
- **Domain objects** - the objects identified from the real world (e.g. bank accounts, tellers, and customers are domain objects).

Types of object

- **Implementation objects** - all objects which are not related to real world entities (e.g. the transaction log is an implementation object).
- **Passive object** - the state of an object will not change unless the object receives a message (e.g. bank account).
- **Active object** - an object that can change its state (e.g. clock, timer).

Basic elements of structural models

Four basic elements of structural models:

- Class
- Attribute
- Operation
- Relationship

Basic element: Class

- Store and manage information in the system.
- A class is a general template that we use to create specific instances, or objects, in the problem domain.
- All objects of a given class are identical in structure and behavior but contain different data in their attributes.
- Two general kinds of classes of interest during analysis:
 - Concrete – application domain classes, used to create objects.
 - Abstract – do not actually exist in the real world, simply useful abstractions.

Basic element: Class

- **Example:** from an employee class and a customer class (concreate classes), we may identify a generalization of the two classes and name the abstract class person.
- A class represents the type of real-word thing, include:
 - domain classes, user-interface classes, data structure classes, file structure classes, operating environment classes, document classes, and various types of multimedia classes.
 - At analysis stage, we are interested only in domain classes.

Basic element: Attribute

- **Attribute** - a piece of information that is relevant to the description of the class within the application domain of the problem.
- An attribute contains information the analyst or user feels the system should keep track of.
- **Example:** An employee class has a possible relevant attribute hair color.
- Only attributes that are important to the task should be included in the class.
 - Primitive or atomic types (i.e., integers, strings, doubles, date, time, Boolean, etc.).

Basic element: Operation

- Operation - The behaviour of an analysis class
- In later phases (e.g. implementation), the operations are converted to methods.
- Only problem domain–specific operations that are relevant to the problem being investigated should be considered.
- For example, it is normally required that classes provide means of:
 - creating instances, deleting instances, accessing individual attribute values, setting individual attribute values, accessing individual relationship values, and removing individual relationship values.

Basic element: Relationship

- There are many different types of relationships that can be defined.
- Three basic categories of data abstraction mechanisms:
 - generalization relationships.
 - aggregation/composition relationships.
 - association relationships.
- Include only relevant relationships.
- Focus on the important dimensions while ignoring nonessential dimensions.

OBJECT IDENTIFICATION

Object Identification

- The four most common approaches to aid the analyst in identifying a set of candidate objects for the structural model:
 - a. textual analysis
 - b. Brainstorming
 - c. common object lists, and
 - d. patterns.
- Use a combination of these techniques to make sure that no important objects and object attributes, operations, and relationships have been overlooked.

Object Identification: Textual Analysis

- Reviewing the use-case diagrams and examining the text in the use-case descriptions to identify potential objects, attributes, operations, and relationships.
- The nouns in the use case suggest possible classes, and the verbs suggest possible operations.
- E.g. in the **Make Old Patient Appt** use case, we can easily identify potential objects:
 - doctor, appointment, patient, office, receptionist, name, address, patient information, payment, date, and time.

Object Identification: Textual Analysis

- Textual Analysis Guidelines:
 - A common or improper noun implies a class of objects.
 - A proper noun or direct reference implies an instance of a class.
 - A collective noun implies a class of objects made up of groups of instances of another class.
 - An adjective implies an attribute of an object.
 - A doing verb implies an operation.
 - A being verb implies a classification relationship between an object and its class.
 - A having verb implies an aggregation or association relationship.
 - A transitive verb implies an operation.
 - An intransitive verb implies an exception.
 - A predicate or descriptive verb phrase implies an operation.
 - An adverb implies an attribute of a relationship or an operation.

Object Identification: Brainstorming

- A discovery technique that has been used successfully in identifying candidate classes.
- A process that a set of individuals sitting around a table suggest potential classes that could be useful.
- Kicked off by a facilitator who asks the set of individuals to address a specific question or statement that frames the session.
 - E.g. using the appointment problem described previously, the facilitator could ask the development team and users to think about **their experiences of making appointments** and to identify candidate classes based on their past experiences.

Object Identification: Common Object Lists

- A list of objects common to the business domain of the system.
- Several categories of objects have been found to help the analyst in creating the list, such as physical or tangible things, incidents, roles, and interactions.
- First look for tangible things in the business domain
 - E.g. books, desks, chairs and office equipment.

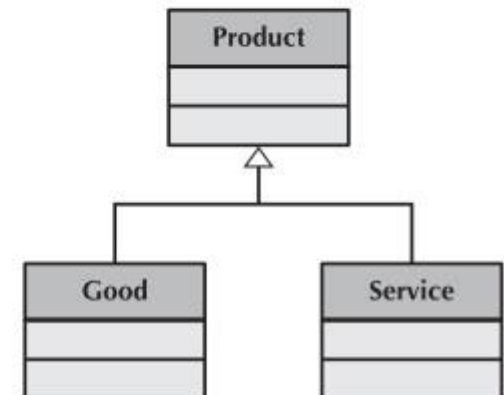
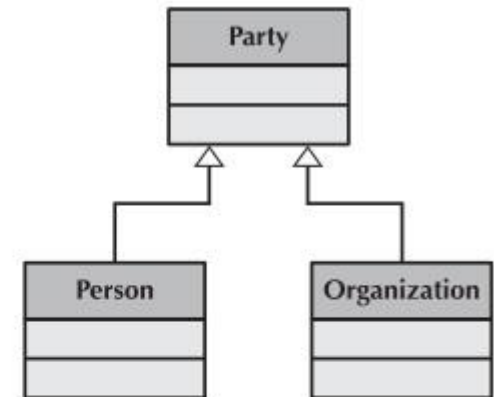
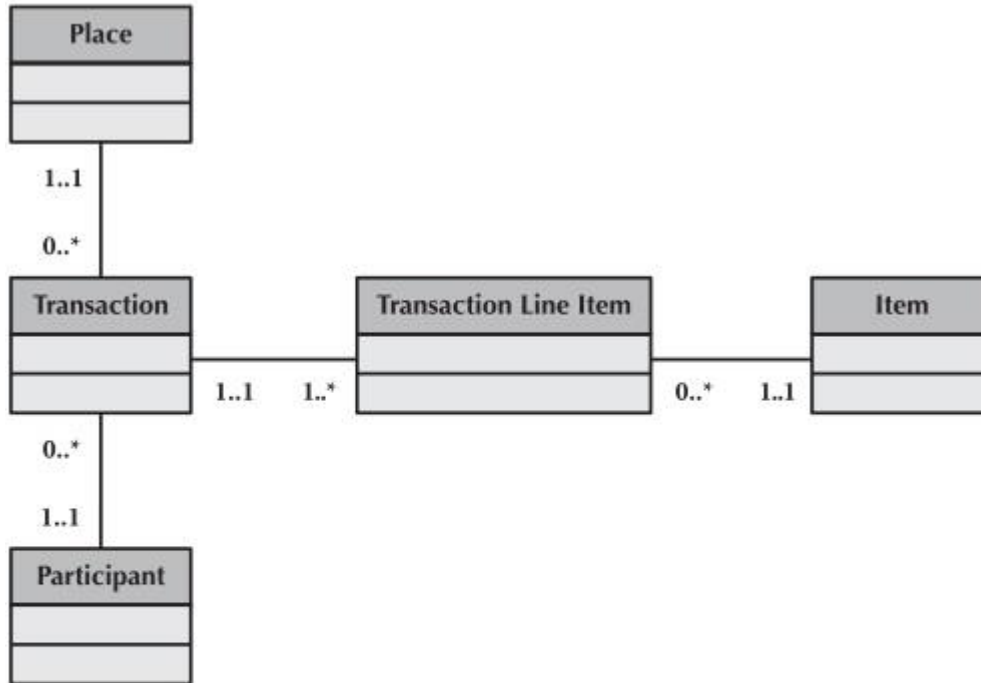
Object Identification: Common Object Lists

- Reviewing the use cases to identify the roles that the people play in the problem, such as doctor, nurse, patient, or receptionist.
- Libraries of reusable objects that have been created for different business domains.
 - For example, with regard to the appointment problem, the Common Open Source Medical Objects could be useful to investigate for potential objects that should be included.

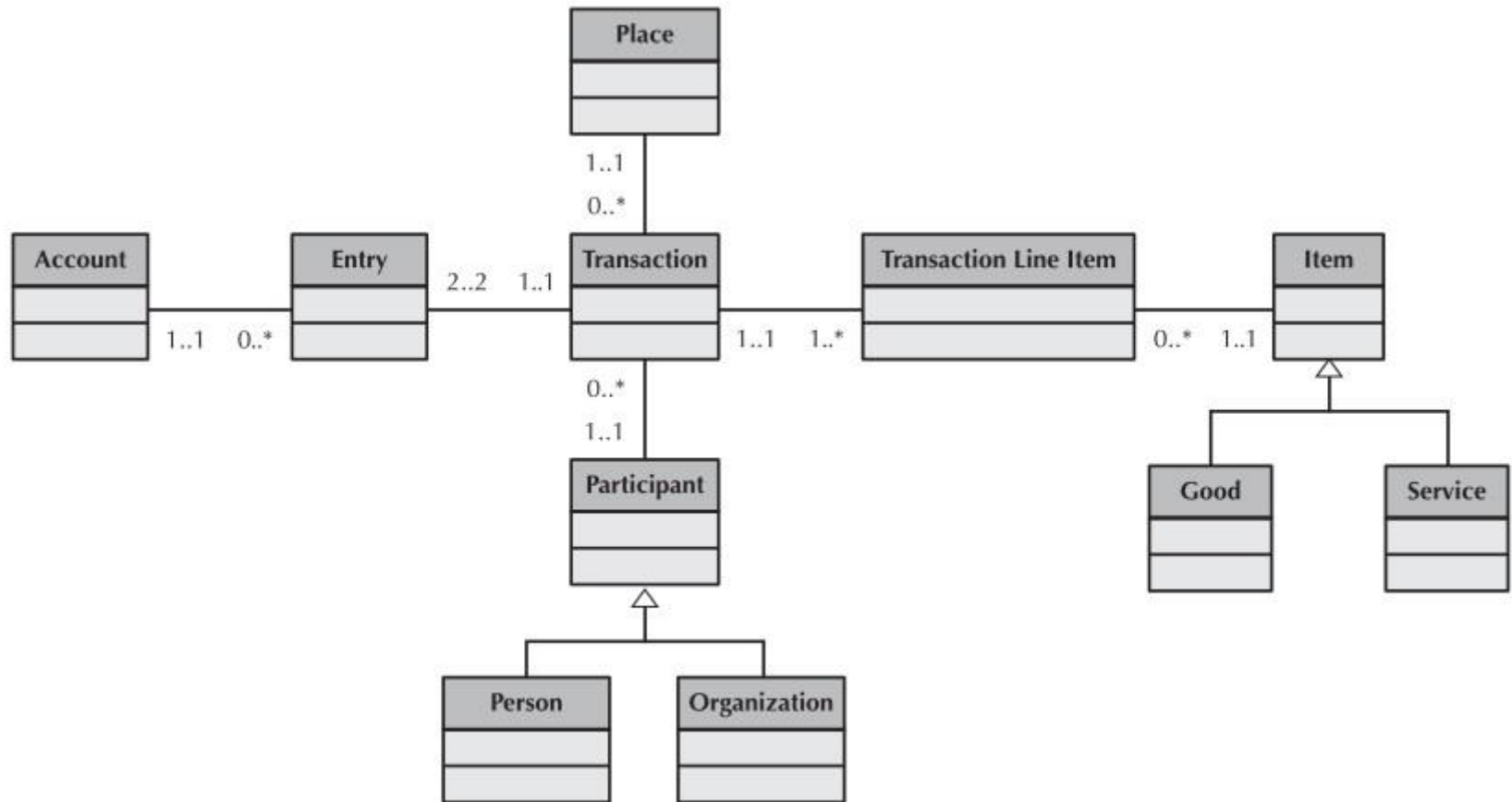
Object Identification: Patterns

- A pattern is simply a useful group of collaborating classes that provide a solution to a commonly occurring problem.
- Patterns provide a solution to commonly occurring problems, they are reusable.
- For example, many business transactions involve the same types of objects and interactions.
- By reusing these existing patterns of classes, we can more quickly and more completely define the system than if we start with a blank piece of paper.

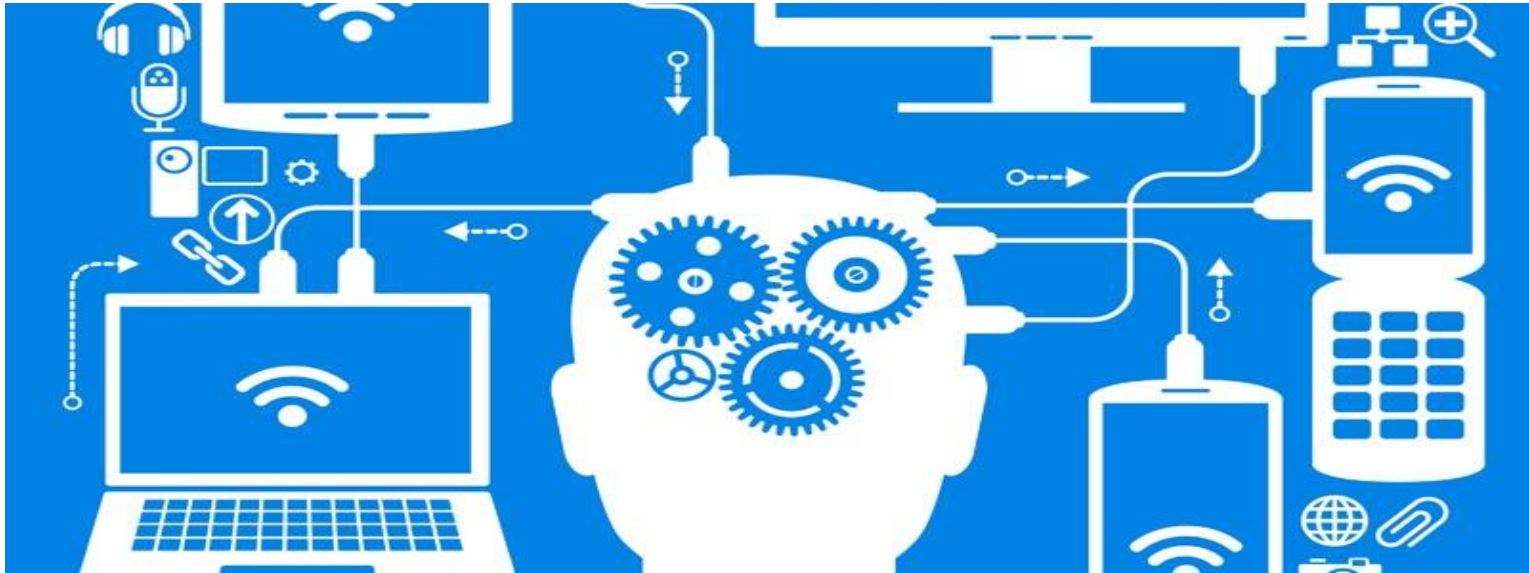
Object Identification: Sample Patterns



Object Identification: Sample Integration of Sample Patterns



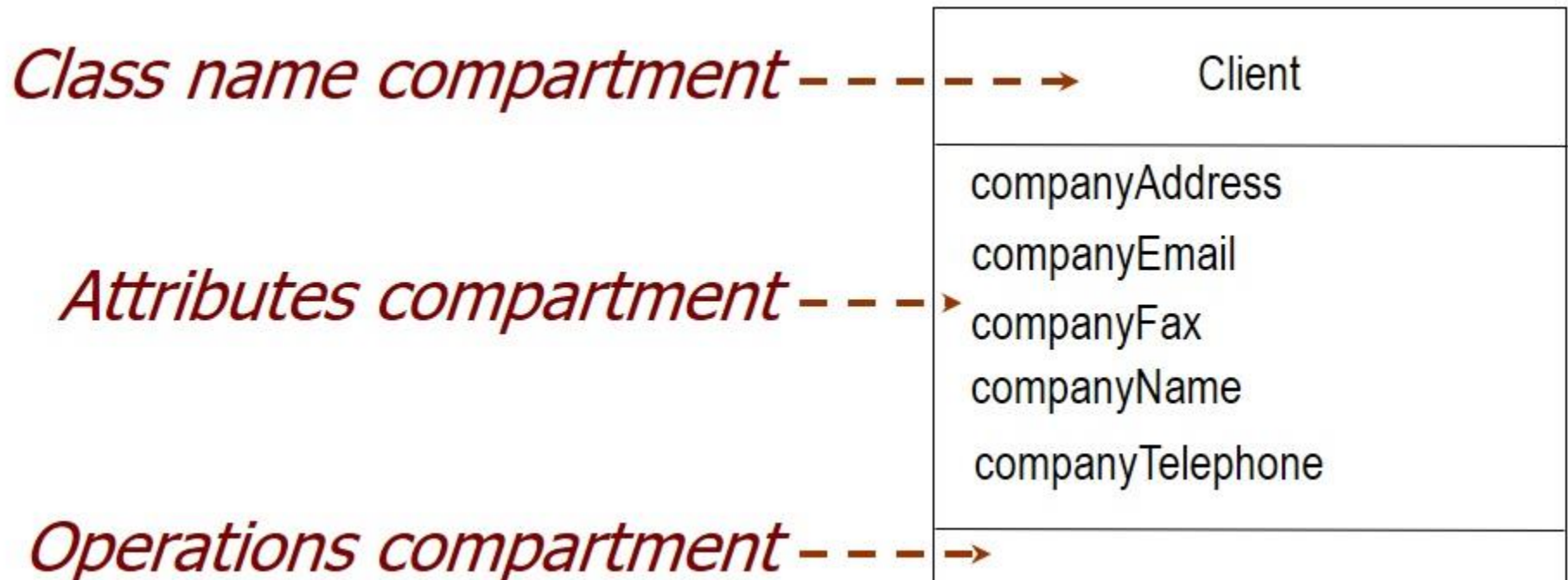
Any question so far..?



UML CLASS DIAGRAMS (NOTATIONS)

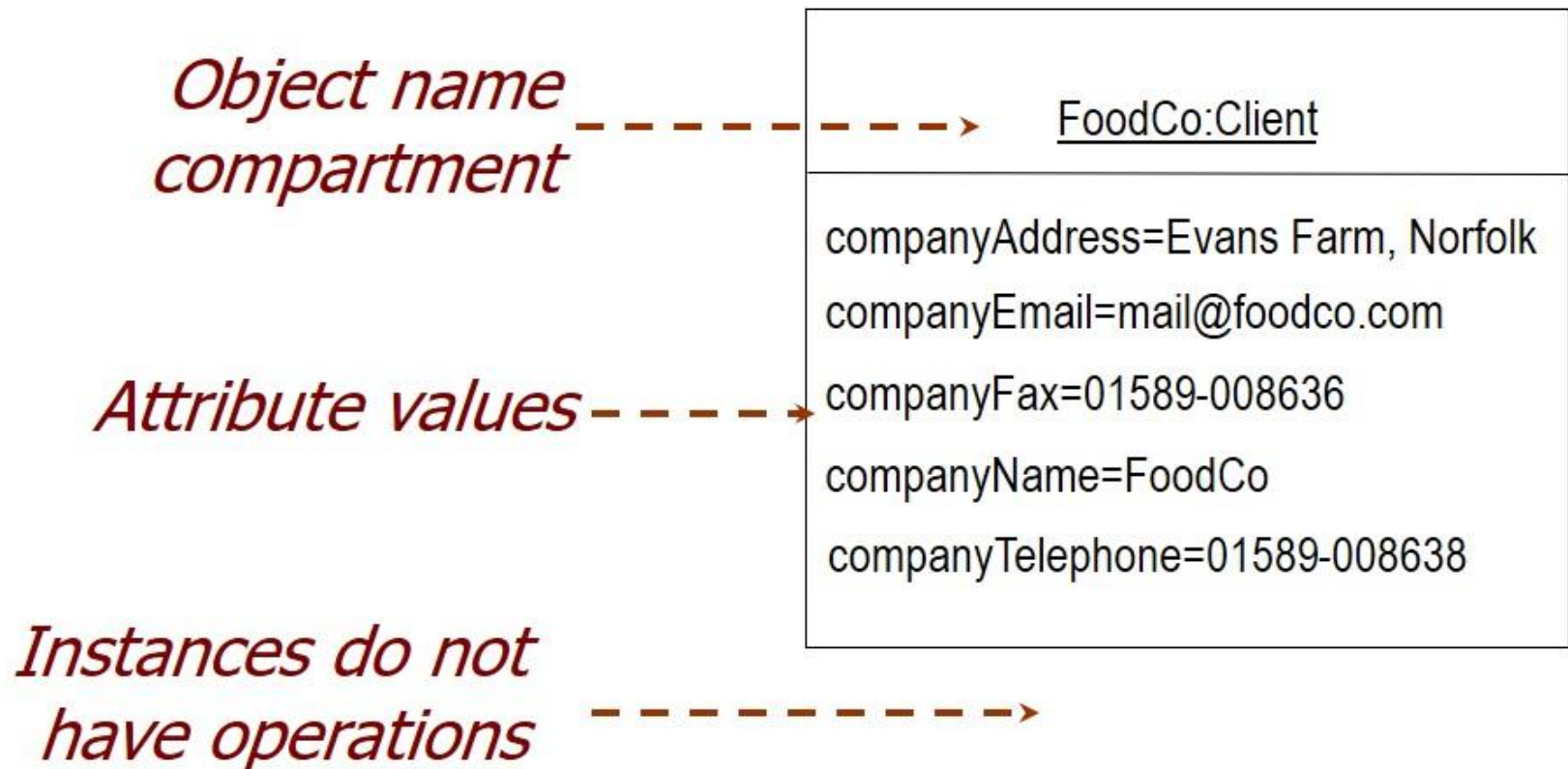
Class Diagram: Class Symbol

- A Class is “a description of a set of objects with similar features, semantics and constraints” (OMG, 2009)



Class Diagram: Instances

- An object (instance) is: “an abstraction of something in a problem domain...”



Class Diagram: Attributes

Attributes are:

- Part of the essential description of a class.
- The common structure of what the class can 'know'.
- Each object has its own *value* for each attribute in its class:
 - *Attribute*="value"
 - companyName=FoodCo

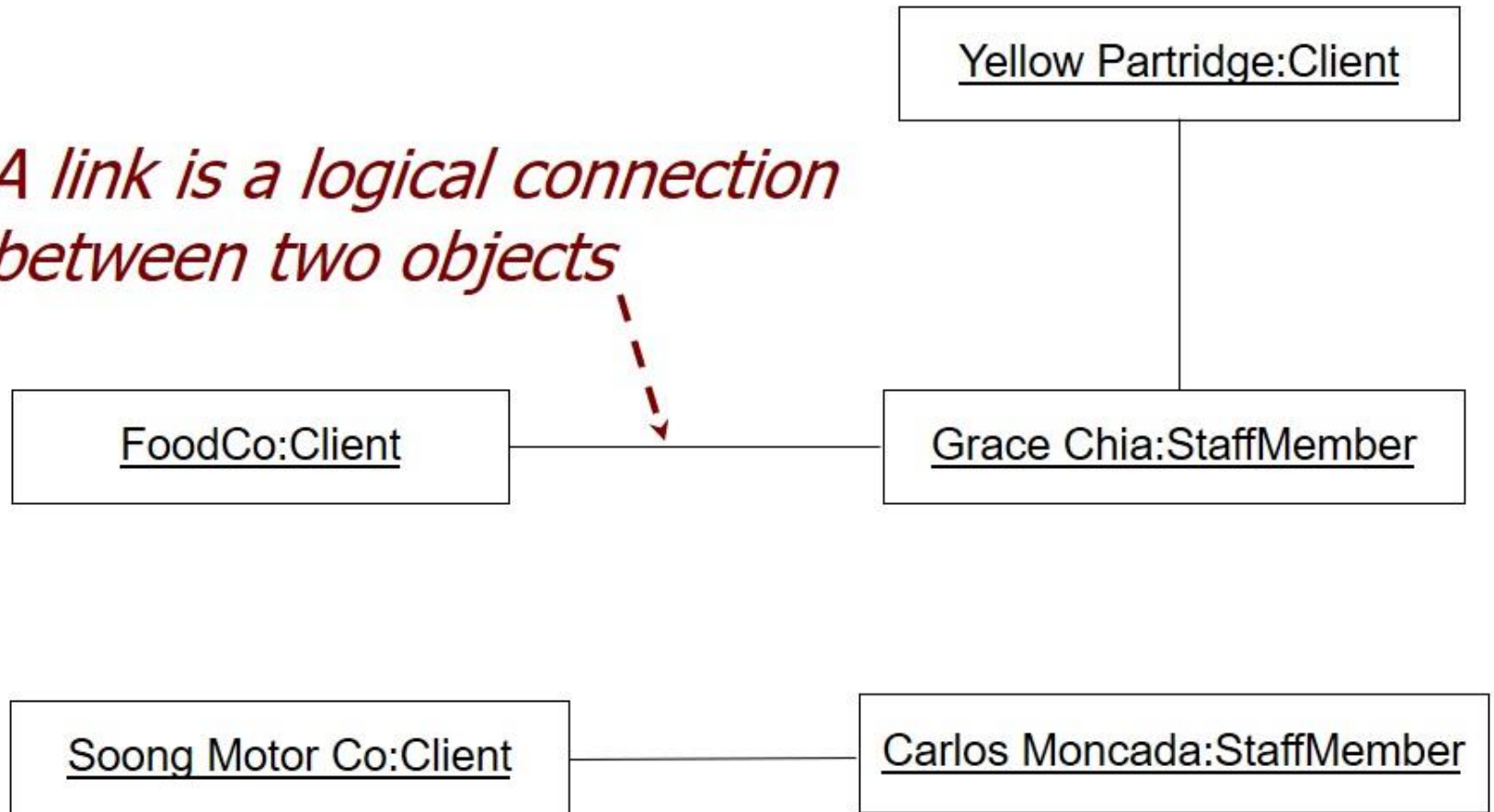
Class Diagram: Associations

Associations represent:

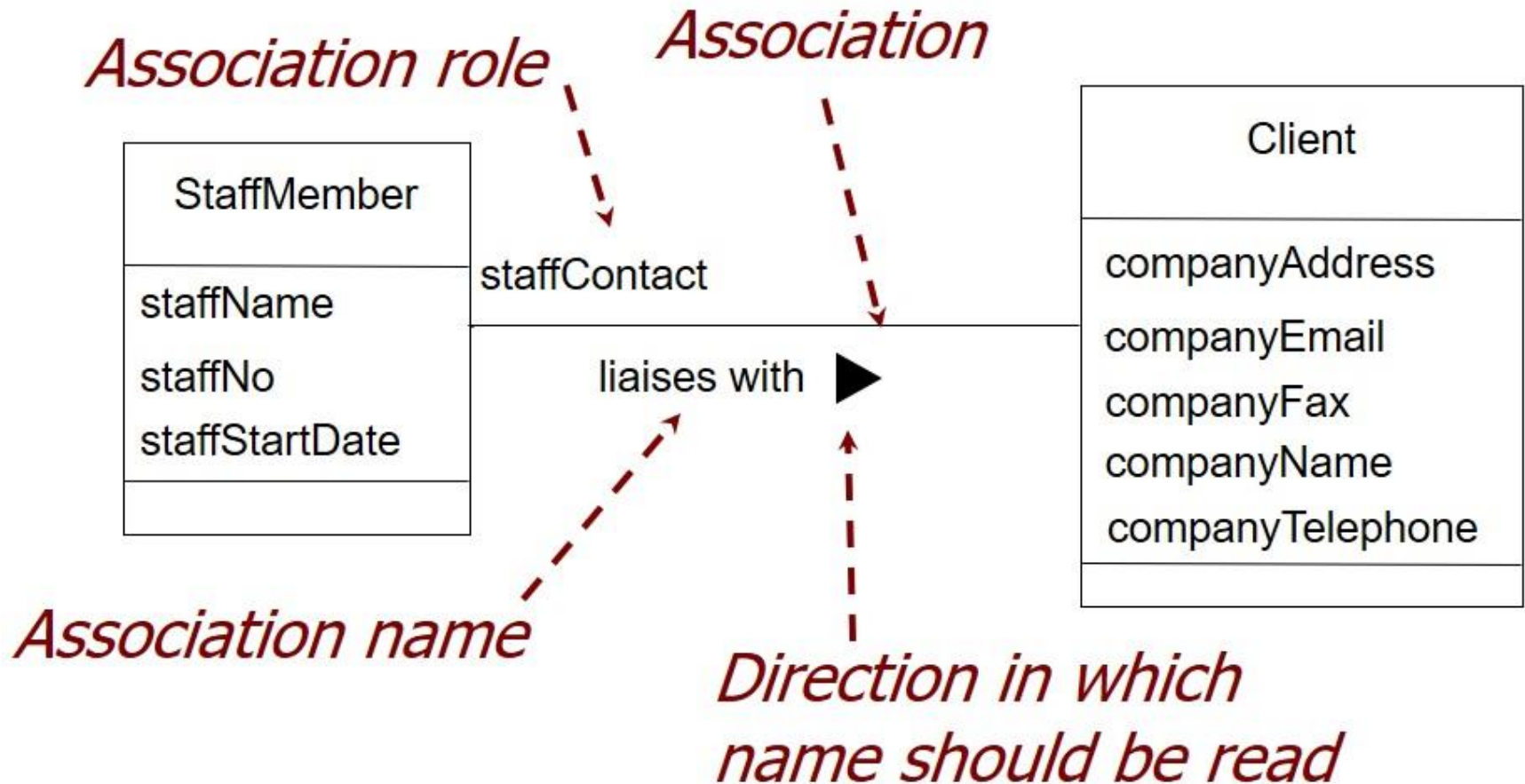
- The possibility of a logical relationship or connection between objects of one class and objects of another.
 - “Grace Chia is the staff contact for FoodCo”
 - *An **employee** object is linked to a **client** object*
- If two objects are linked, their classes are said to have an association.
- A line is drawn and labelled with either the name of the relationship or the roles that the classes play in the relationship.

Class Diagram: Links

*A link is a logical connection
between two objects*



Class Diagram: Associations

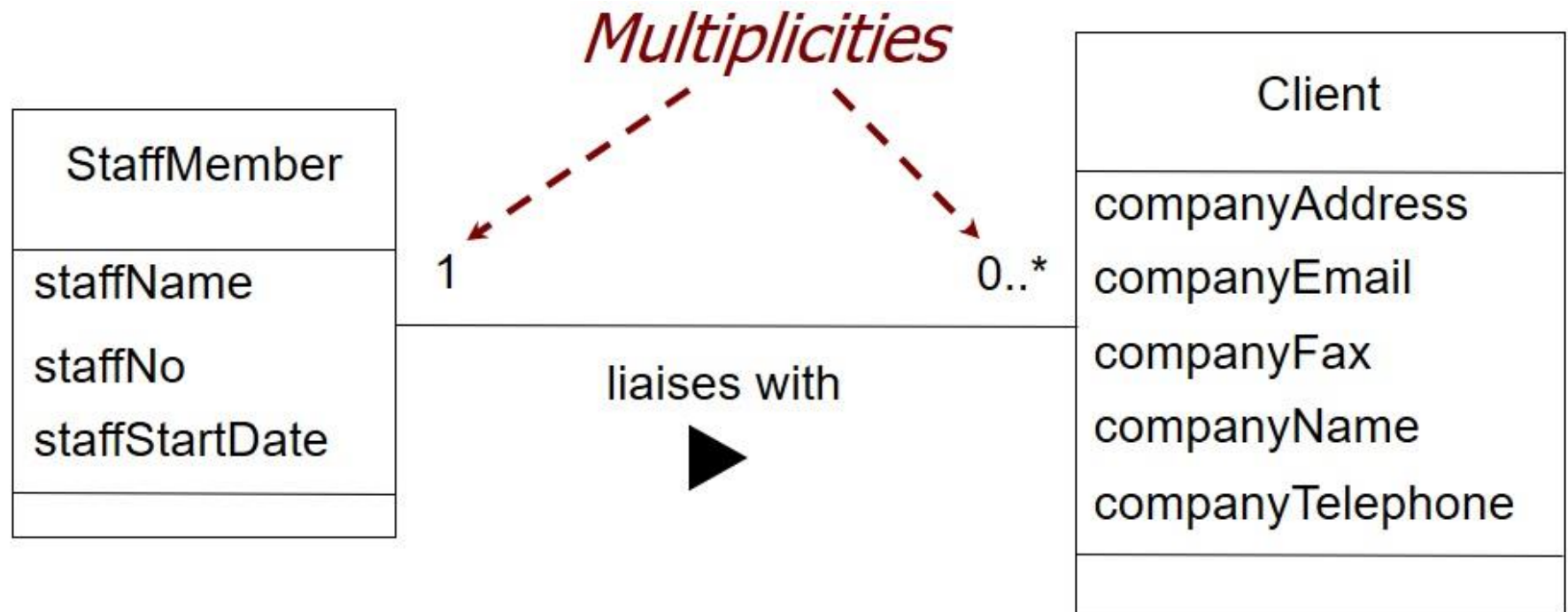


Class Diagram: Multiplicity







- Associations have multiplicity: the range of permitted cardinalities of an association.
- Represent *enterprise (or business) rules*.
- Numbers are placed on the association path to denote the **minimum** and **maximum** instances (minimum number.. maximum number).
- These always come in pairs:
 - Associations must be read separately from both ends
 - Each **bank customer** may have 1 or more **accounts**
 - Every **account** is for 1, and only 1, **customer**

Class Diagram: Multiplicity (Notation)

- Exactly one staff member liaises with each client.
- A staff member may liaise with zero, one or more clients.



Class Diagram: Multiplicity (Example)

Exactly one	1	 <pre> classDiagram Department "1" -- "1" Boss </pre>	A department has one and only one boss.
Zero or more	0..*	 <pre> classDiagram Employee "1" -- "0..*" Child </pre>	An employee has zero to many children.
One or more	1..*	 <pre> classDiagram Boss "1" -- "1..*" Employee </pre>	A boss is responsible for one or more employees.
Zero or one	0..1	 <pre> classDiagram Employee "1" -- "0..1" Spouse </pre>	An employee can be married to zero or one spouse.
Specified range	2..4	 <pre> classDiagram Employee "1" -- "2..4" Vacation </pre>	An employee can take from two to four vacations each year.
Multiple, disjoint ranges	1..3,5	 <pre> classDiagram Employee "1" -- "1..3,5" Committee </pre>	An employee is a member of one to three or five committees.

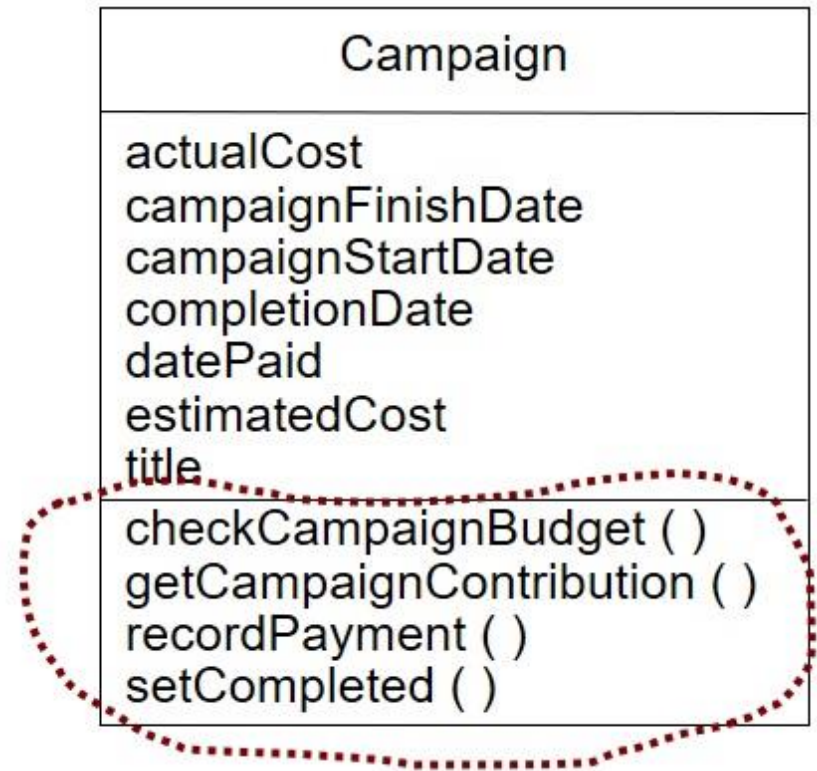
Class Diagram: Operations

Operations are:

- An essential part of the description of a class.
- The common behaviour shared by all objects of the class.
- Services that objects of a class can provide to other objects.
- There are four kinds of operations that a class can contain:
constructor, query, update, and destructor.

Class Diagram: Operations (Notation)

- Operations describe actions or functions that the instances of a class can do:
 - Set or reveal attribute values
 - Perform calculations
 - Send messages to other objects
 - Create a new instance
 - Delete an instance



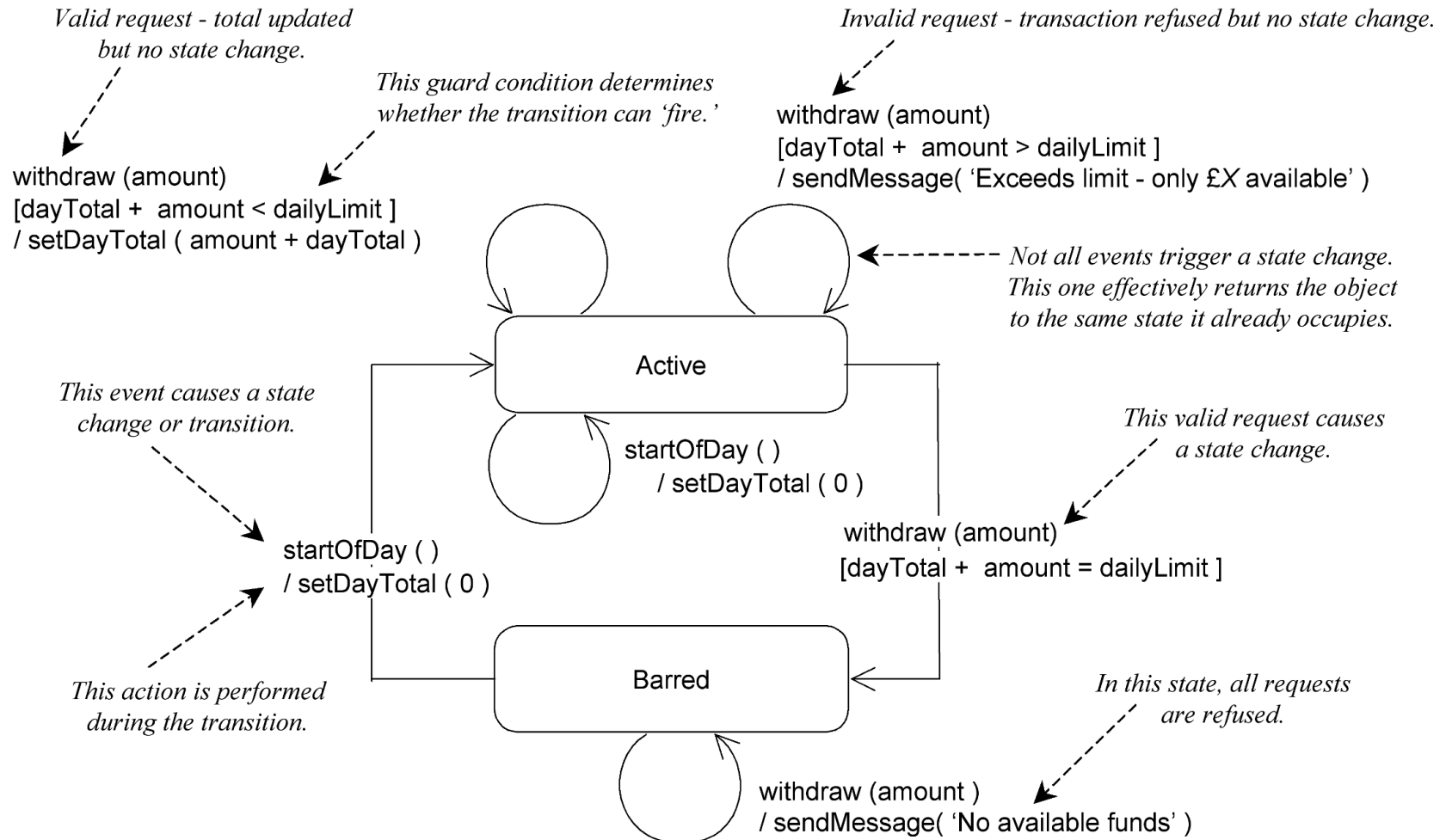
Object State

- An object's state is related to its attributes, its links and its operations.
- Current state is an encapsulation of the value of attributes and links.
- State constrains behaviour - it determines whether or not an operation can fire.
- Executing an operation often causes a change of state.

Object State

- In the ATM example on the following slide, an account object responds differently according to the current value of:
 - `dayTotal` (amount withdrawn so far today)
 - `dailyLimit` (total that can be withdrawn)
- Together, these define the object's state: `Active` or `Barred`.
- The current state determines whether a `withdraw` operation can be successful.

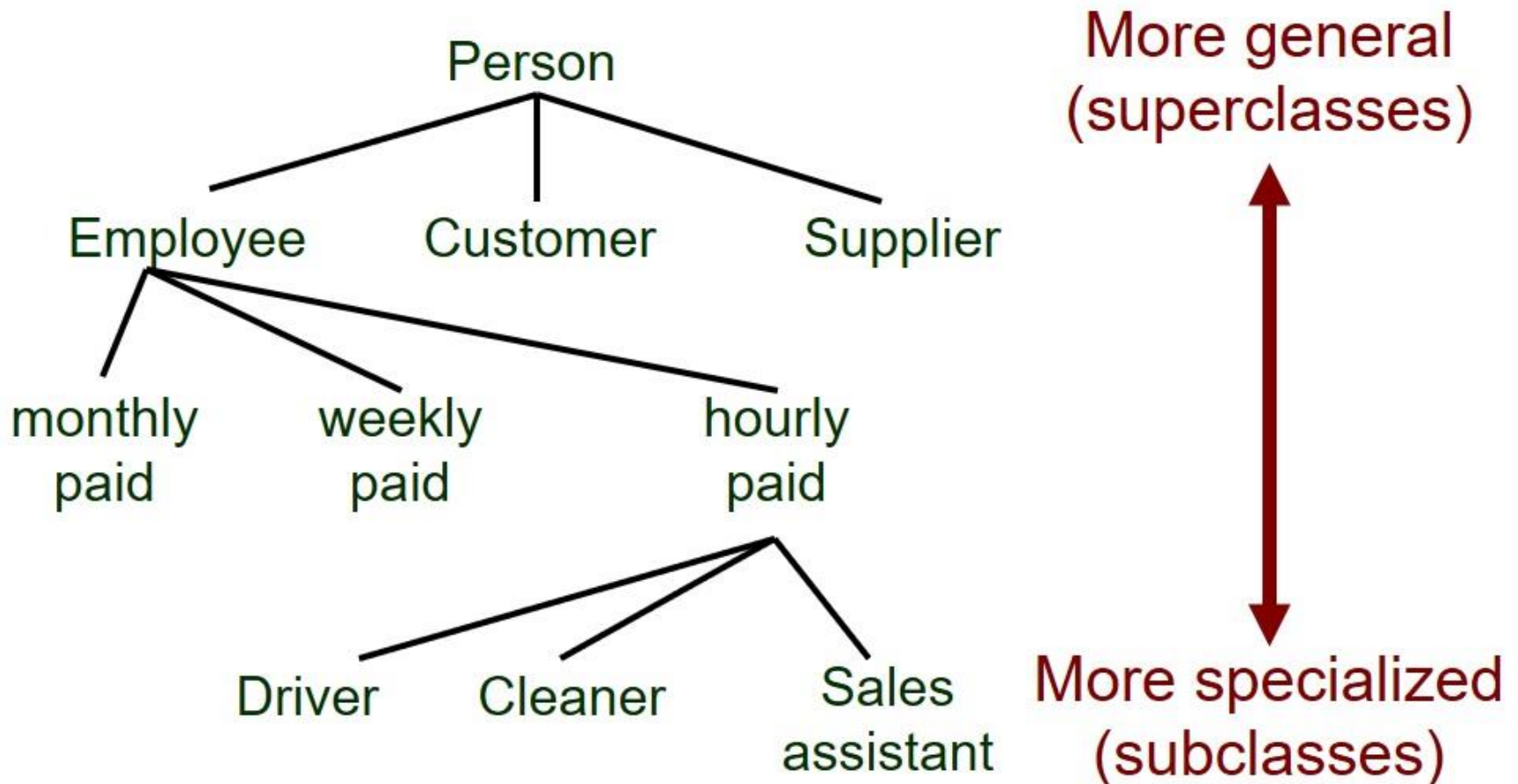
Object State - Example



Generalization (a-kind-of relationship)

- Create classes (subclasses) that inherit basic attributes and operations of other classes (superclass).
 - E.g. An employee is a-kind-of person
- Specialization uncovers additional classes by allowing new subclasses to be created from an existing class.
 - E.g. An employee class can be specialized into a secretary class and an engineer class

Generalization (a-kind-of relationship)



Generalization (a-kind-of relationship)

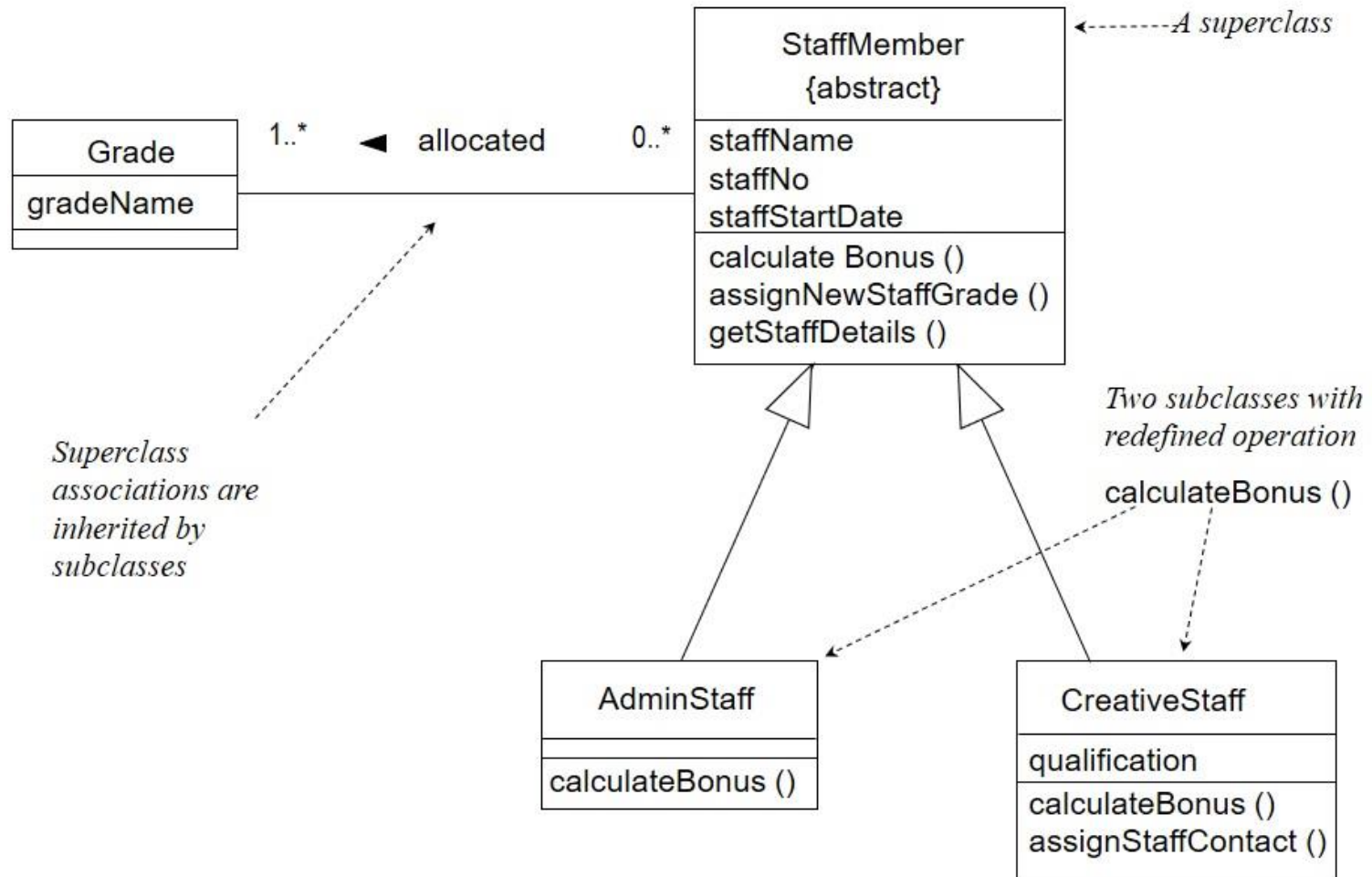
- Focus on the properties that make each class unique by allowing the similarities to be factored into superclasses.
- Add generalization structures when:
 - Two classes are similar in most details, but differ in some respects
 - May differ:
 - In behaviour (operations or methods)
 - In data (attributes)
 - In associations with other classes

Generalization (Example)

- CreativeStaff is a-kind-of StaffMember
- AdminStaff is a-kind-of StaffMember

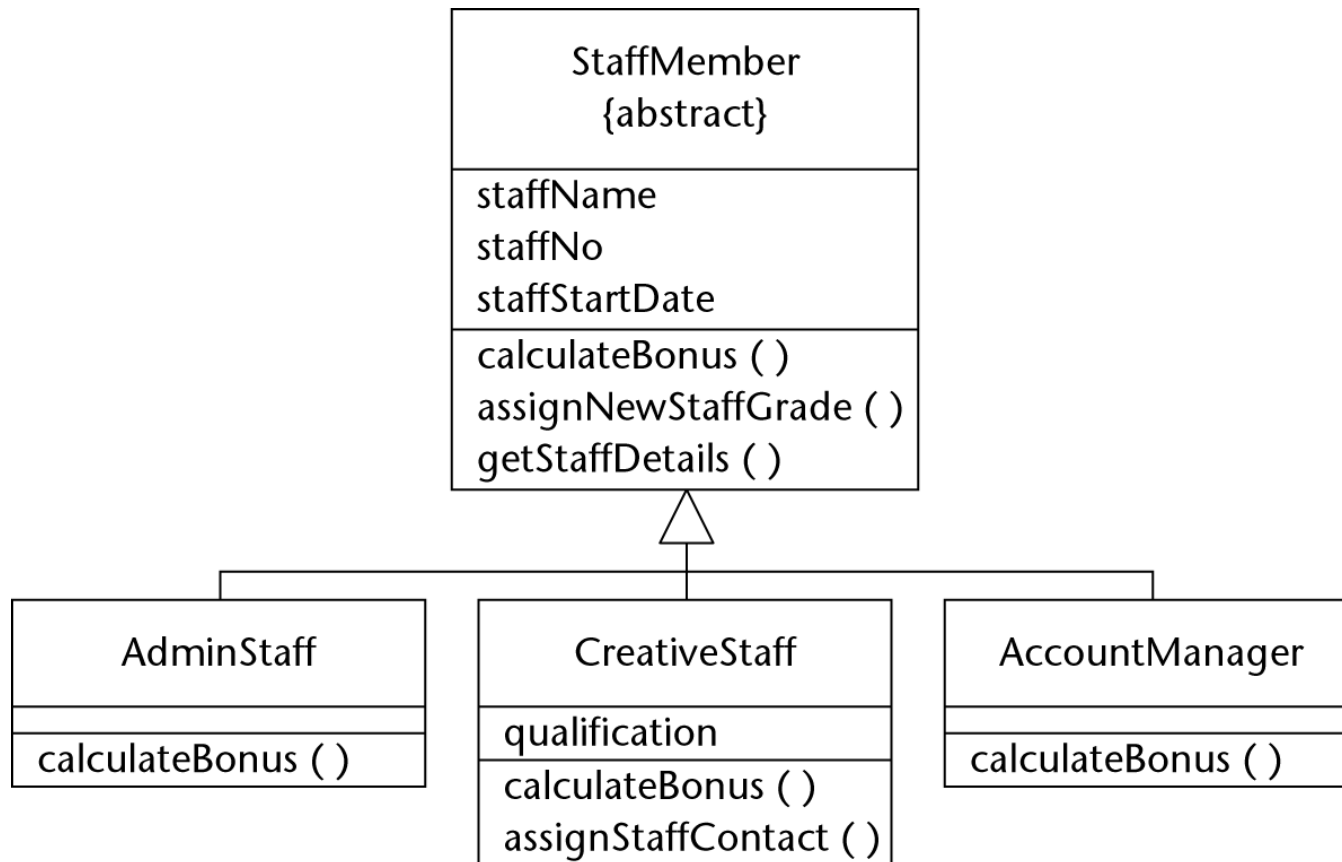
Creative	Have qualifications recorded Can be client contact for campaign Bonus based on campaigns they have worked on
Admin	Qualifications are not recorded Not associated with campaigns Bonus not based on campaign profits

Generalization (Example)



Generalization - Reuse

- A generalization hierarchy can be reused in other contexts, within the current application.

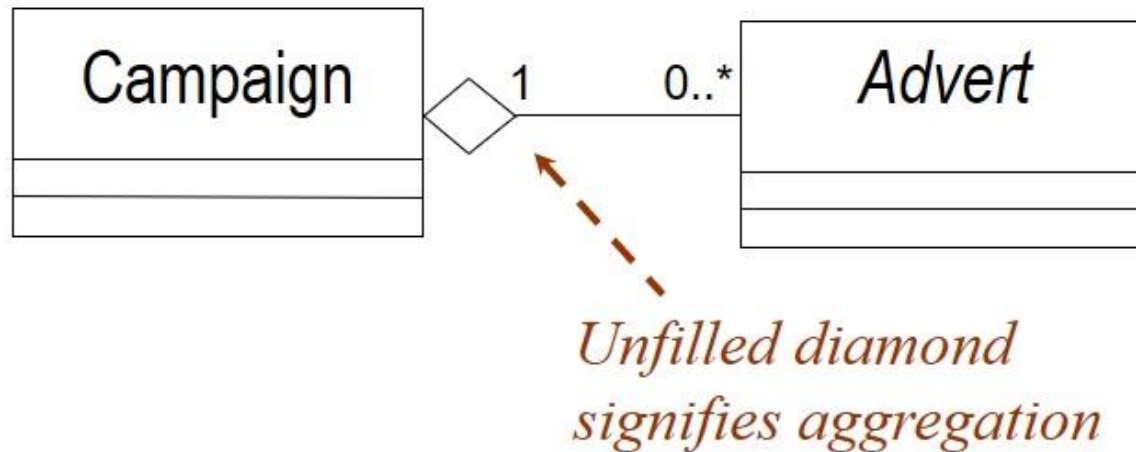


Aggregation and Composition

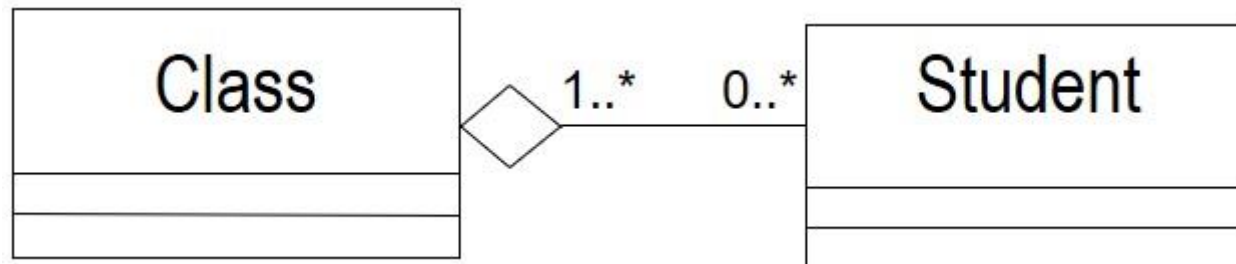
- Special types of association, both sometimes called whole-part (has-a, a-part-of or has-parts) semantic relationship to represent the aggregation abstraction.
 - Example: a door is a-part-of a car, an employee is a-part-of a department
- **Aggregation** is essentially any whole-part relationship (weak type).
 - May survive without the other
- 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.

Example of Aggregation

- A campaign is made up of adverts:

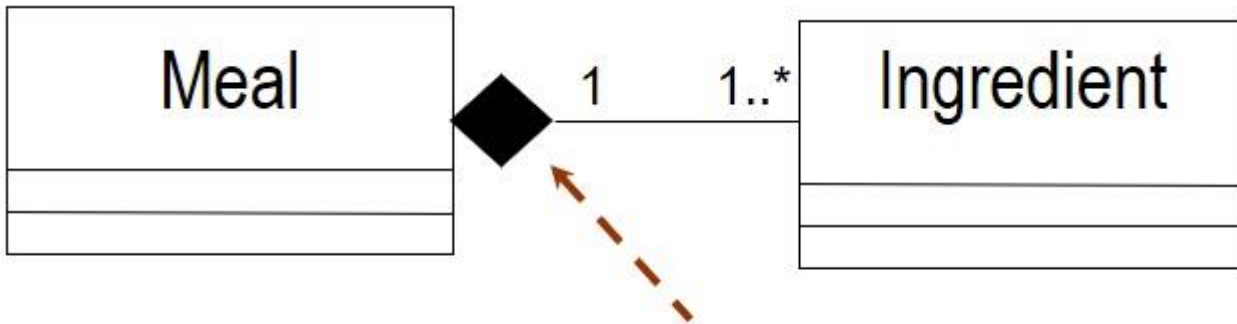


- An everyday example: A class has many zero to students.



Example of Composition

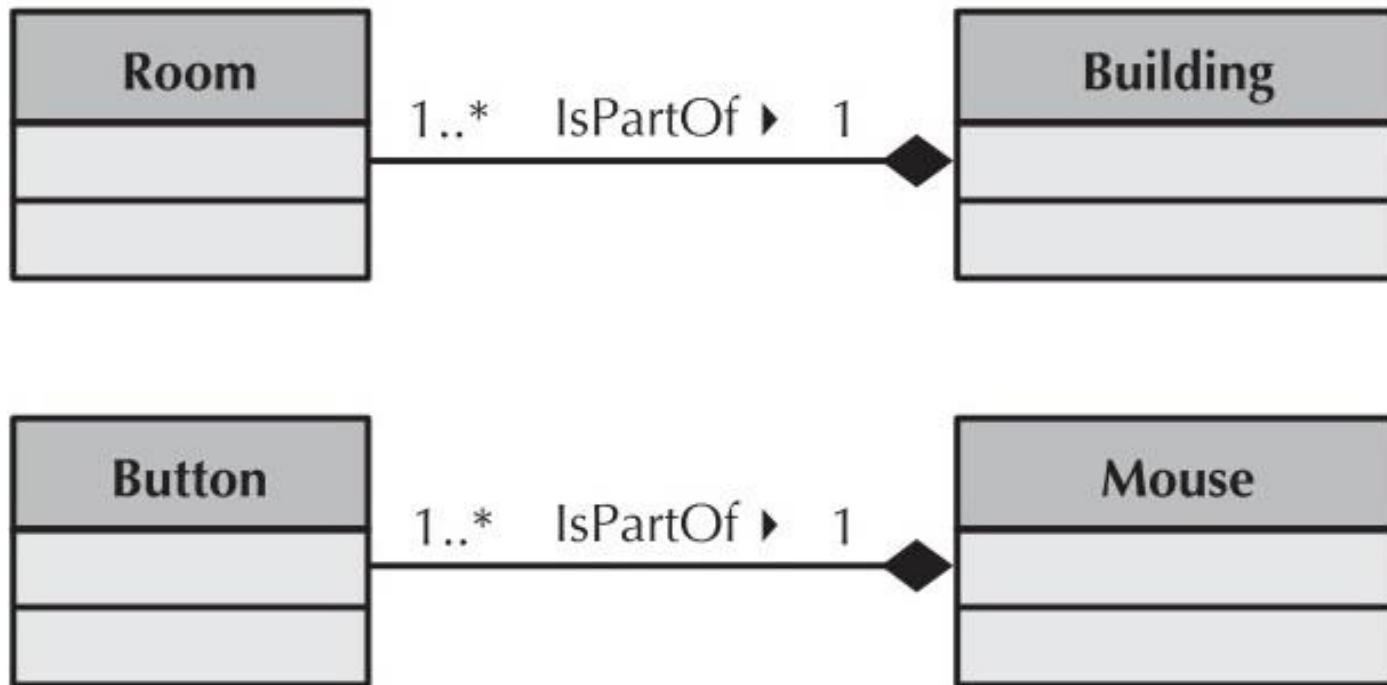
- Another everyday example:



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.
 - One can't exist without the others.

Sample Composition Associations

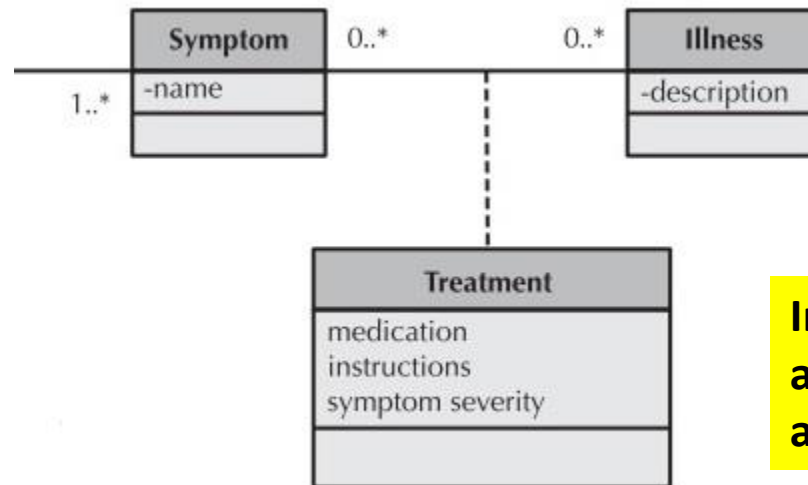


Association Class

- There are times when a relationship itself has associated properties
 - especially when its classes share a **many-to-many** relationship.
 - break into a set of one-to-many relationships
- An association class is a cross between an association and a class.
- Warning: You cannot attach an association class to more than one association.

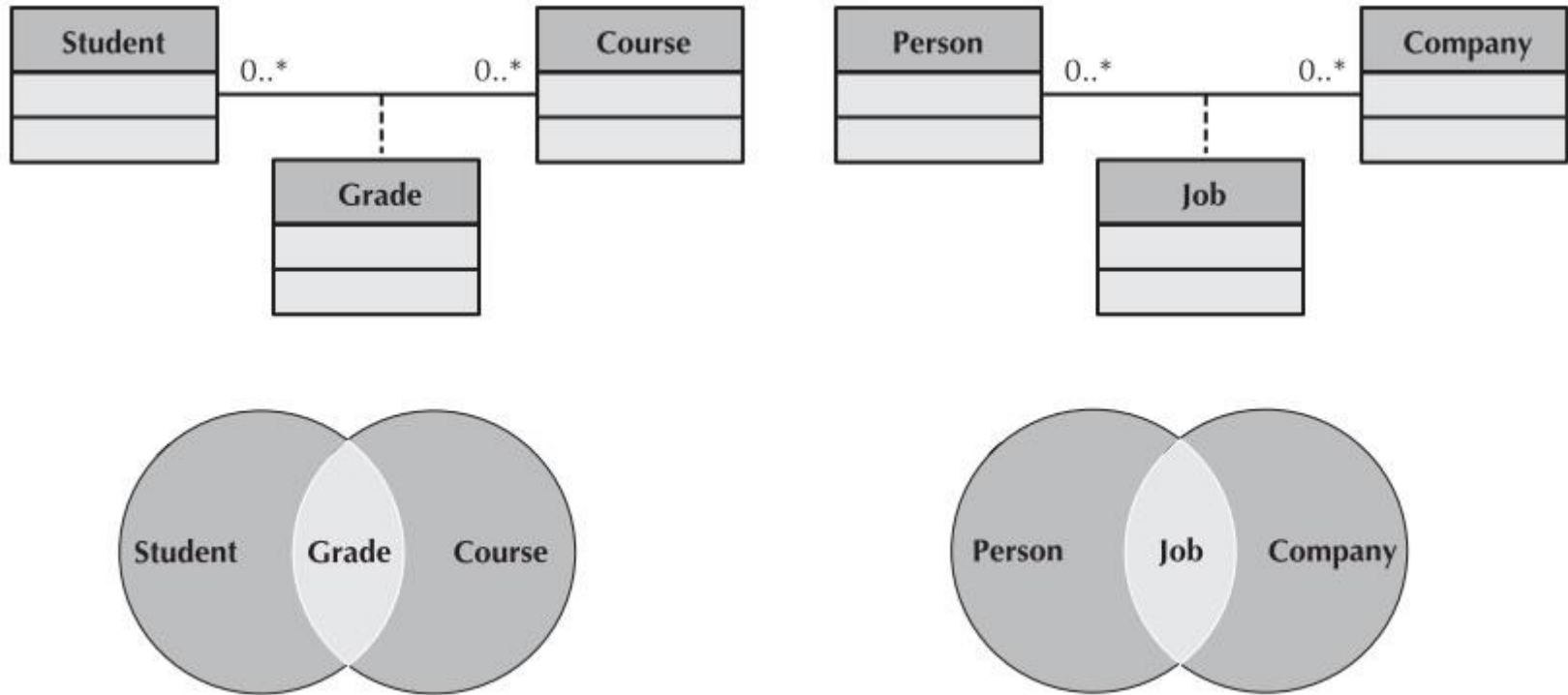
Association Class (Notation)

- It is shown as a **rectangle attached by a dashed line to the association path**, and the rectangle's name matches the label of the association.
- Think about the case of capturing information about illnesses and symptoms:



Include additional attributes to describe an association

Association Class (Examples)



A Grade is an intersection of the **Student** and **Course** classes

A job may be viewed as the intersection between a **Person** and a **Company**

Dependency

- A dependency is a **using relationship** that states that:
 - a change in specification of one thing (independent thing) may affect another thing that uses it (dependent thing)
 - But not necessarily the reverse
- It is rendered as a dashed directed line.

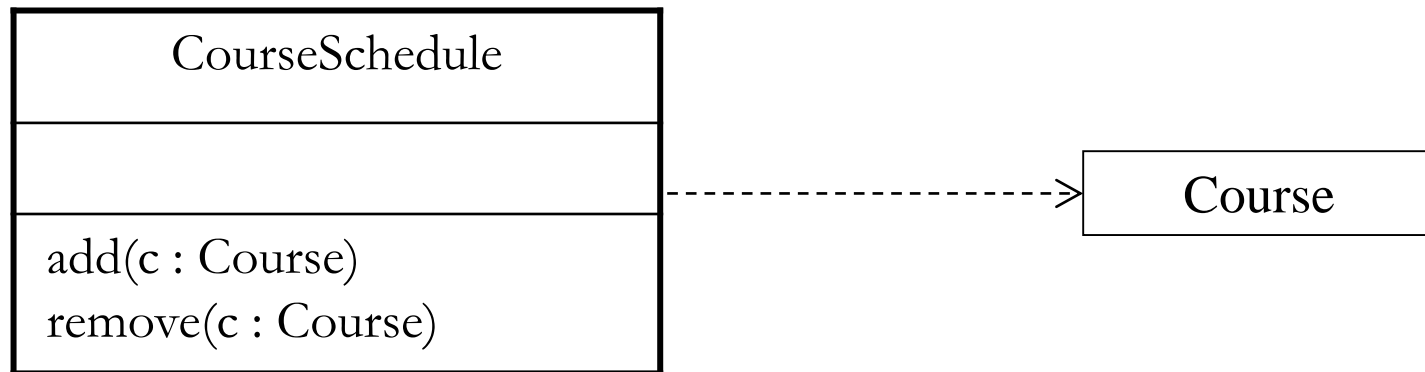
dependent -----> *independent*

Dependency (cont.)

- Used in the context of classes to show that:
 - one class uses another class as an argument in the signature of an operation
 - if the used class changes, the operation of the other class may be affected
- The most common kind of dependency relationship is the connection between a class that only **uses another class as a parameter** to an operation.

Example of Dependency

- There's a dependency from CourseSchedule to Course, as Course class is used as a parameter in both the add() and remove() operations.



Constraints and Notes

- Constraints allowing you to add new rules or modify existing ones.
- 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.

Example of Constraints and Notes

Figure 2.17. Example of constraints

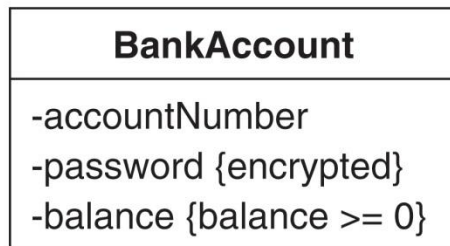


Figure 2.18. Complex constraints

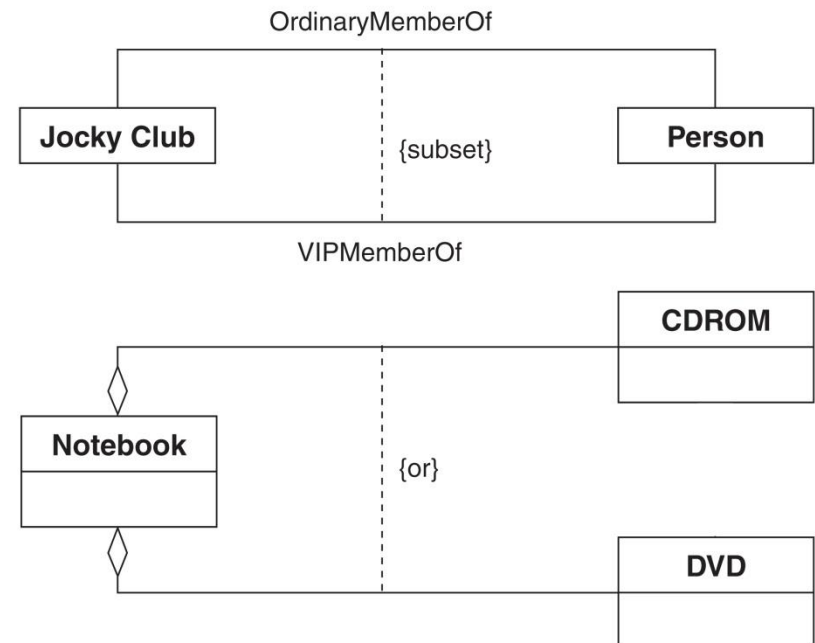
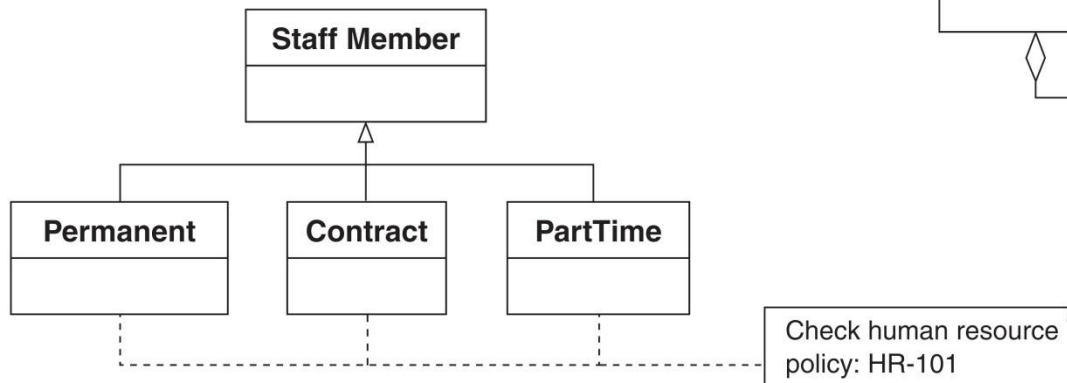
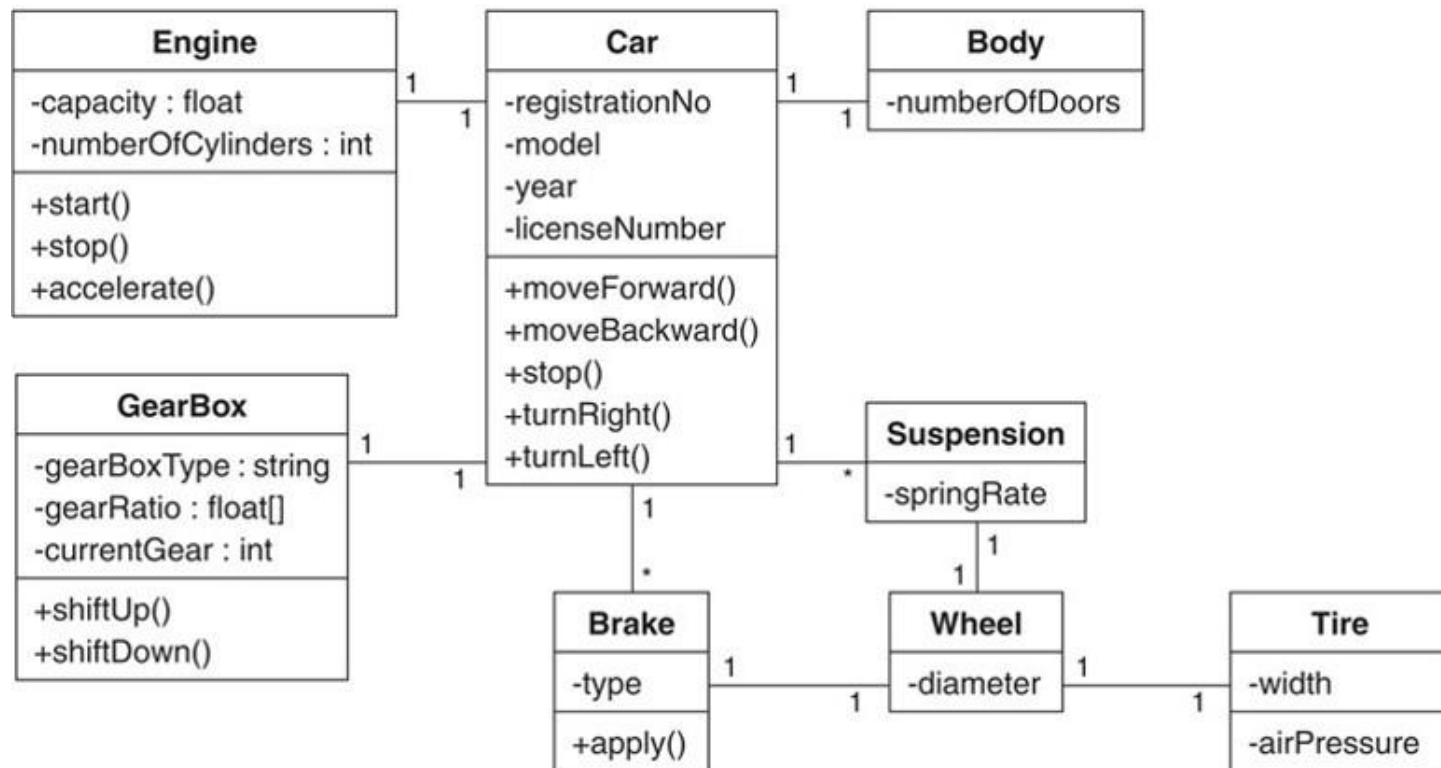


Figure 2.19. Note annotation

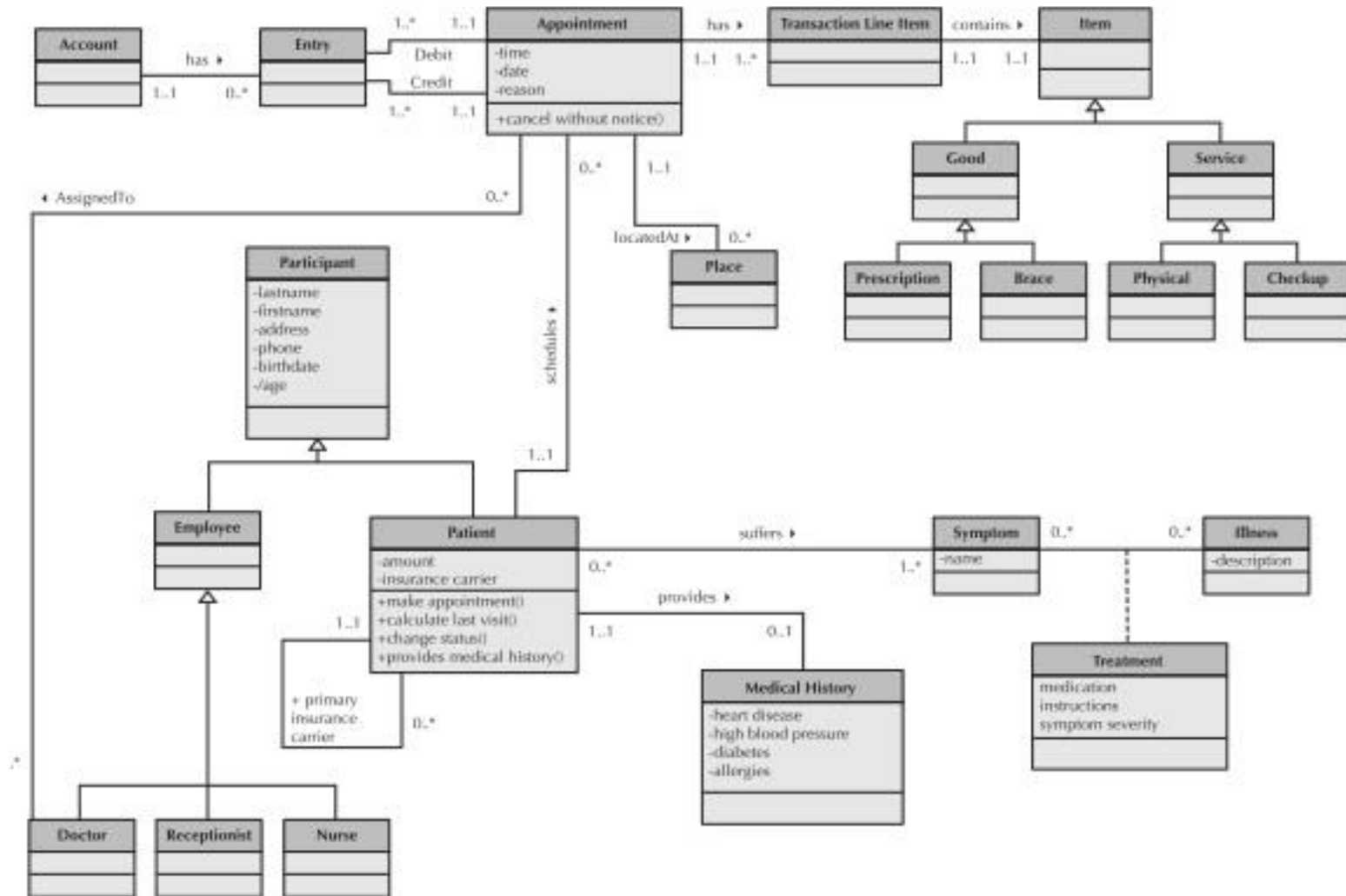


Sample Class Diagrams

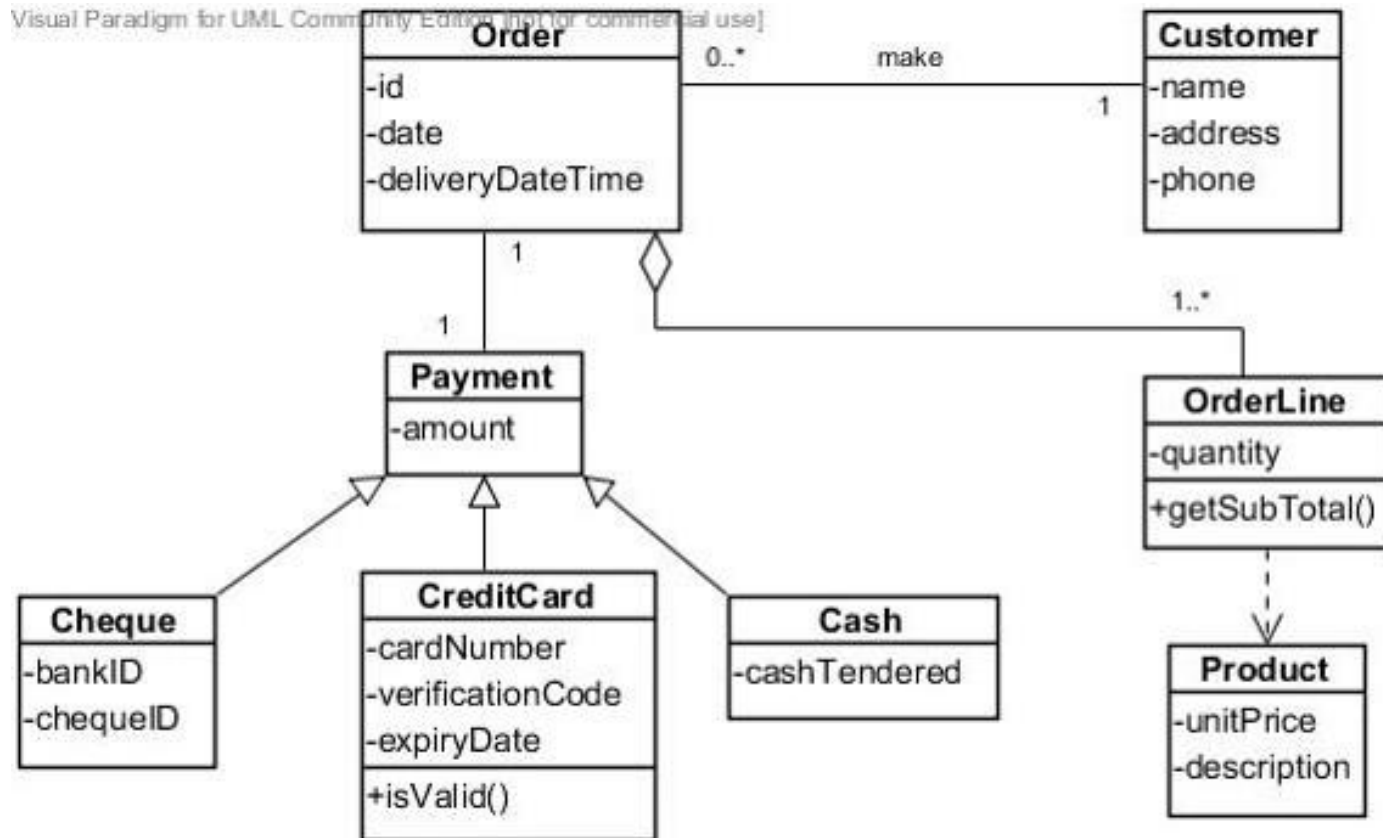
- A class diagram shows classes and various relationships in which they are involved.
- Example: Class diagram of a car.



Sample Class Diagram: Appointment System



Sample Class Diagrams: A Sales Order System



Key points

- We analyse requirements
 - There may be repetition
 - Some parts may already exist as standard components
 - Use cases give little information about structure of software system
- A structural model is a formal way of representing the objects that are used and created by a software system.
- Objects can be identified using textual analysis, brainstorming, common object lists, and patterns.

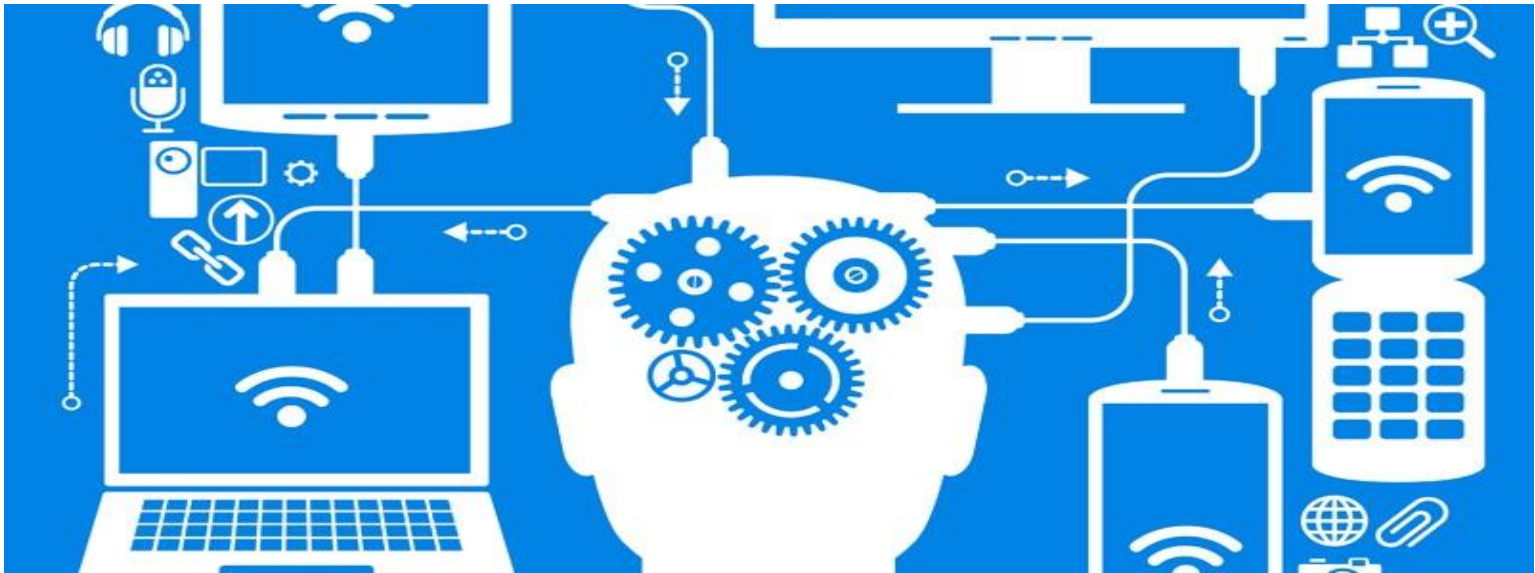
Key points

- Four basic elements of structural models:
 - Class
 - Attribute
 - Operation
 - Relationship
- An object's state is related to its attributes, its links and its operations.

References

- Alan Dennis, Barbara Haley Wixom & David Tegarden. 2015. Systems Analysis and Design with UML, 5th edition, Wiley.
- Simon Bennett, Steve McRobb & Ray Farmer. 2010. Object Oriented Systems Analysis and Design using UML 4th Edition, McGraw-Hill.

In the next lecture..



Lecture 6: UML Class Diagrams (Part 2)