

## 3.2 Class Diagrams in Analysis

Subject/Topic/Focus:

- ❑ Introduction to Classes

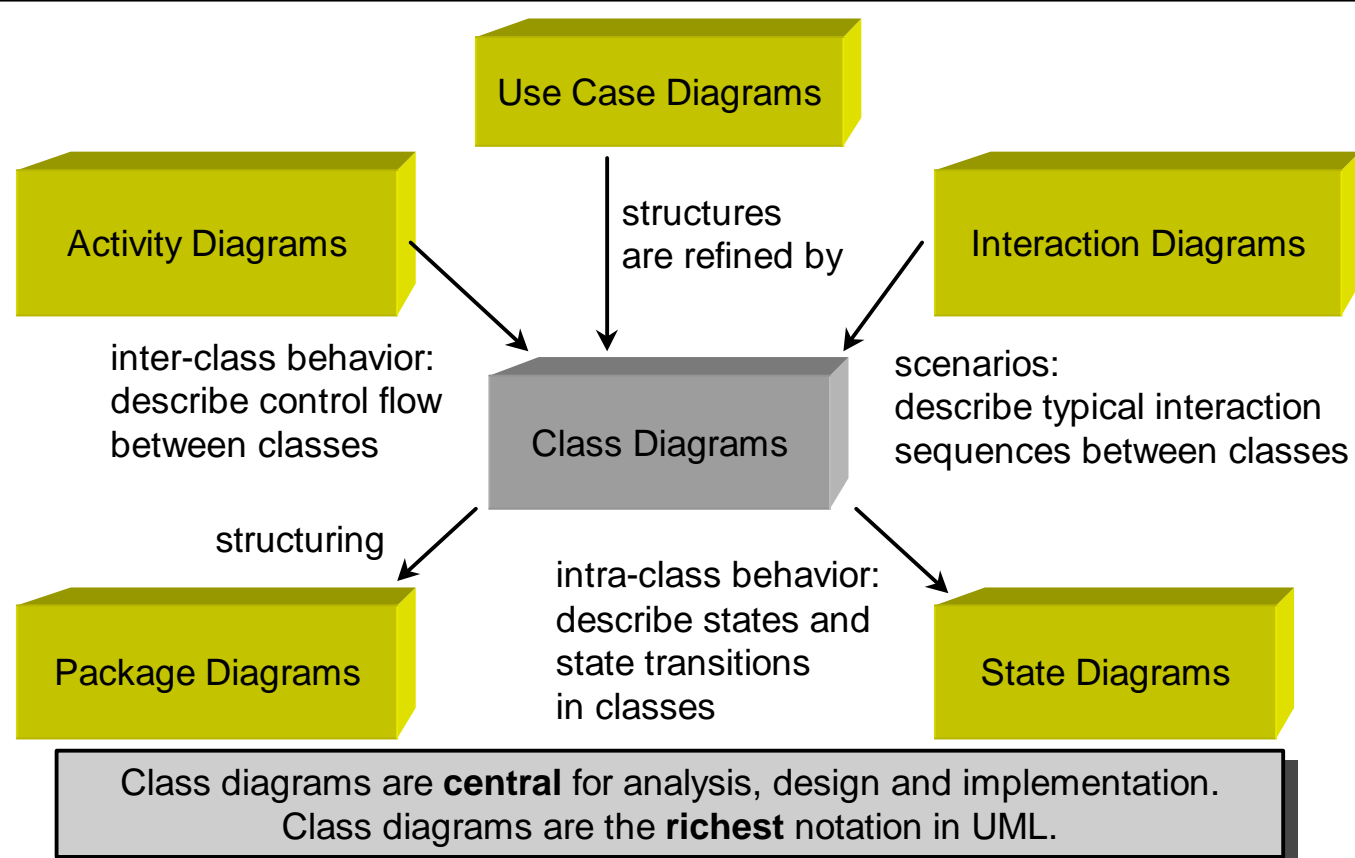
Summary:

- ❑ Conceptual Modeling
- ❑ Notation:
  - Classes
  - Associations: Multiplicity, Roles, Aggregation, Composition
  - Generalization
  - Objects
- ❑ Analysis Process

Literature:

- ❑ [Fowler97]
- ❑ [Booch98]

## Role of Class Diagrams



## Conceptual Modeling of Classes

### Goal:

Conceptual perspective

- ❑ represents the concepts belonging to the classes
- ❑ provides language independence

### Resources

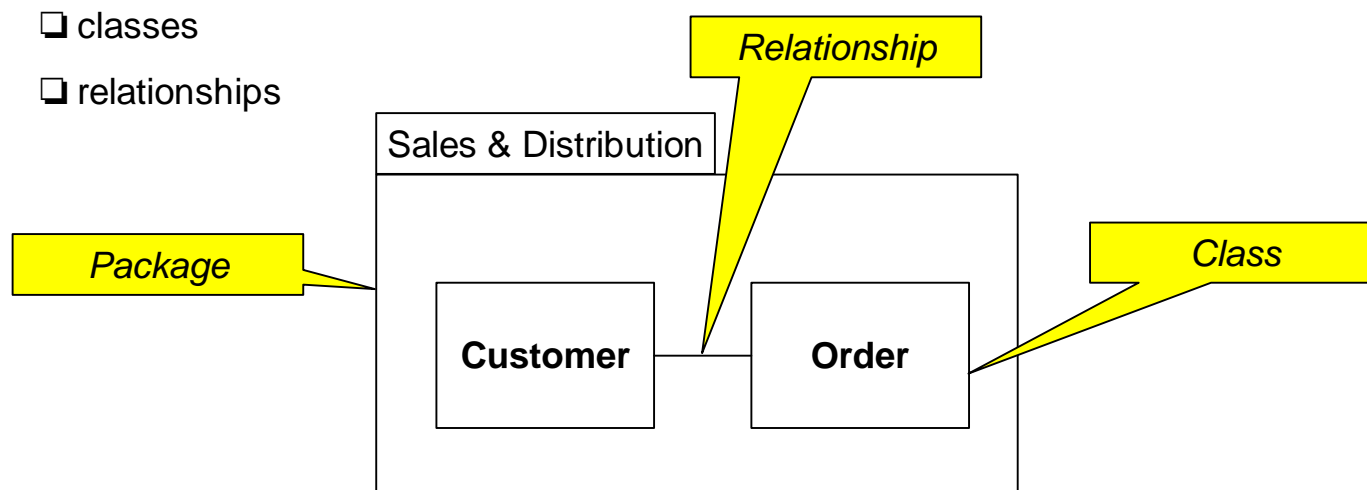
- ❑ class diagram
  - class
  - relationship
- ❑ object diagram
  - object (as an instance of a class)
  - reference

## Class Diagram

A **class diagram** is a graphical representation of a **static view** on declarative, static elements.

Class diagrams contain:

- ❑ packages
- ❑ classes
- ❑ relationships

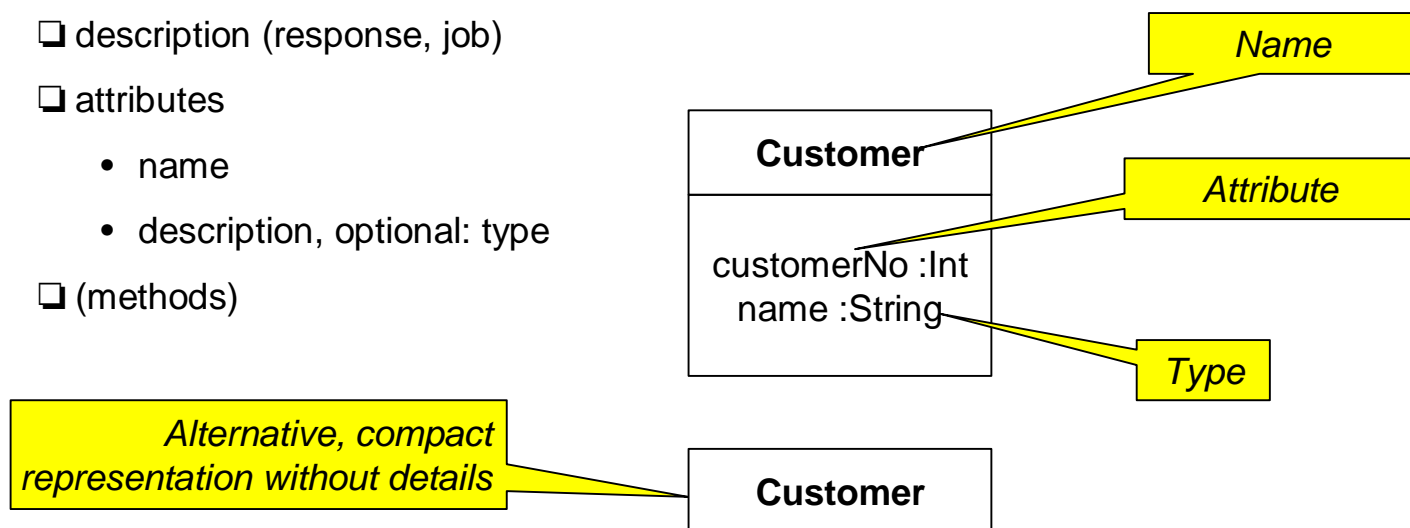


## Class in the Conceptual Class Diagram

A **class** is the description of a set of objects having similar attributes, operations, methods, relationships and behavior.

A class has

- name
- description (response, job)
- attributes
  - name
  - description, optional: type
- (methods)

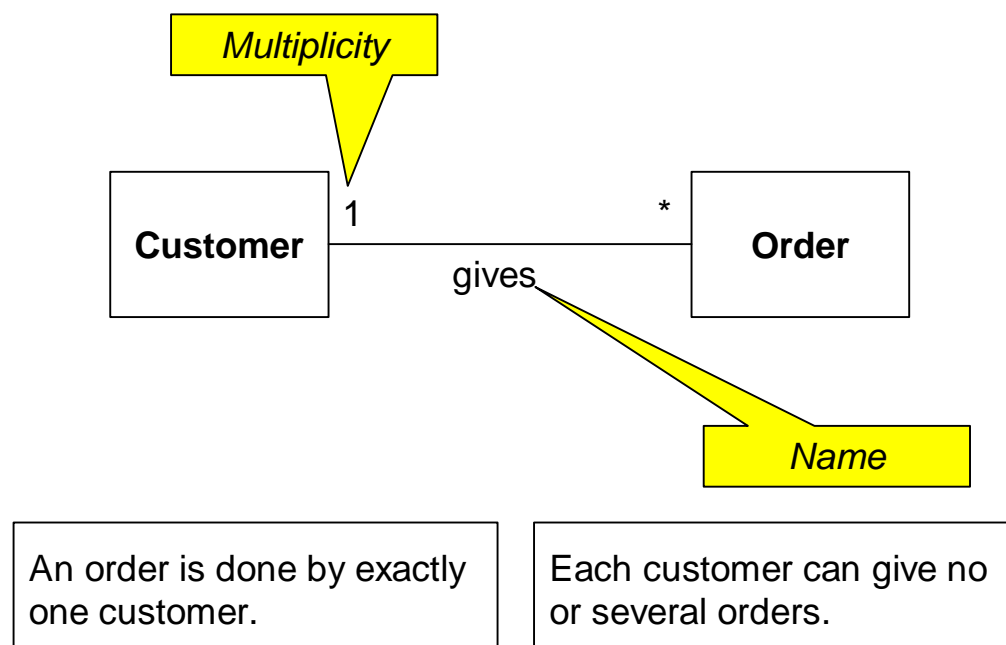


## Association

An **association** is a semantic relationship between classes which concerns the connection (e.g., references) between its instances.

**Notation:**

- name
- multiplicity
- description (extracted from use cases)



## Multiplicity

The multiplicity of an association defines the valid range of values for the number of objects taking part in the association.

### Notation:

- number
- number..number
- number..\*
- no numbrt := \*
- \*

### Example:

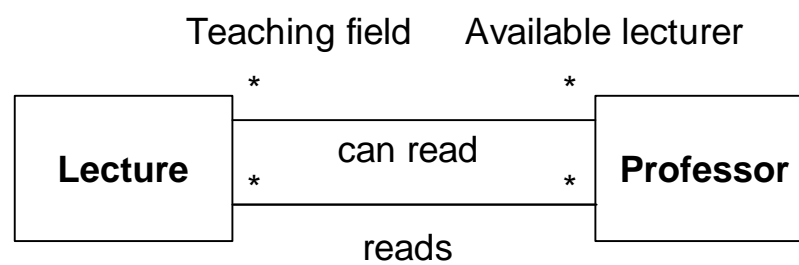
1	3,4	
0..1	1..2	0..1,4
1..*	2..*	
		*

Equivalent to 0..\*

## Role

A **role** in an association is a name describing the participation of the class in the association more exactly.

### Notation:



Teaching offer    Lecturer

Role

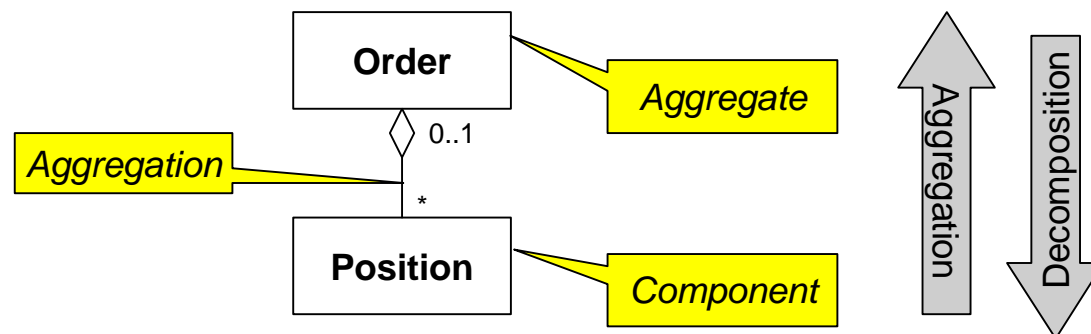
## Aggregation / Composition

An **aggregation** resp. **composition** is a semantic relationship between an **aggregate** A and a **component** B. "B is a part of A".

Possible properties of the aggregation:

- dependent existence of the component (component may not exist without aggregate and dies with it)
- exclusive aggregation (component can only be component of a single aggregate)
- transitivity of the component relationship

Notation:



Class Diagrams in Analysis

3.2.9

## Generalization

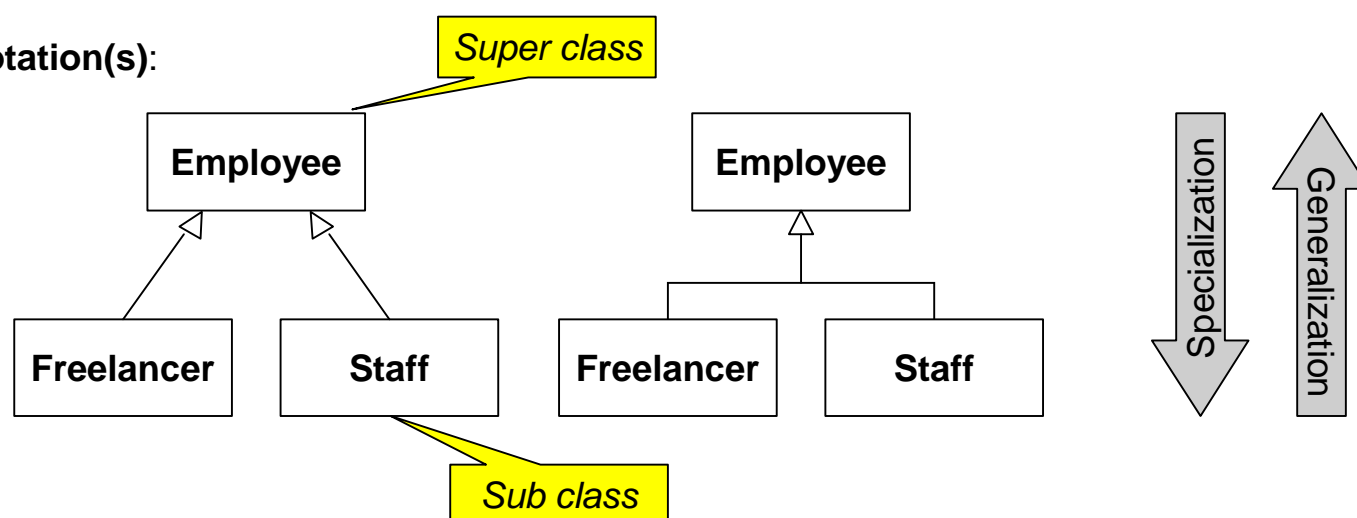
**Generalization** is a semantic relationship between a general concept A (**super class**) and a more special concept B (**sub class**).

„B specializes A“, „each B is an A“

For classes: **Inheritance** builds a **taxonomy**.

[Possible properties of generalization follow later]

Notation(s):

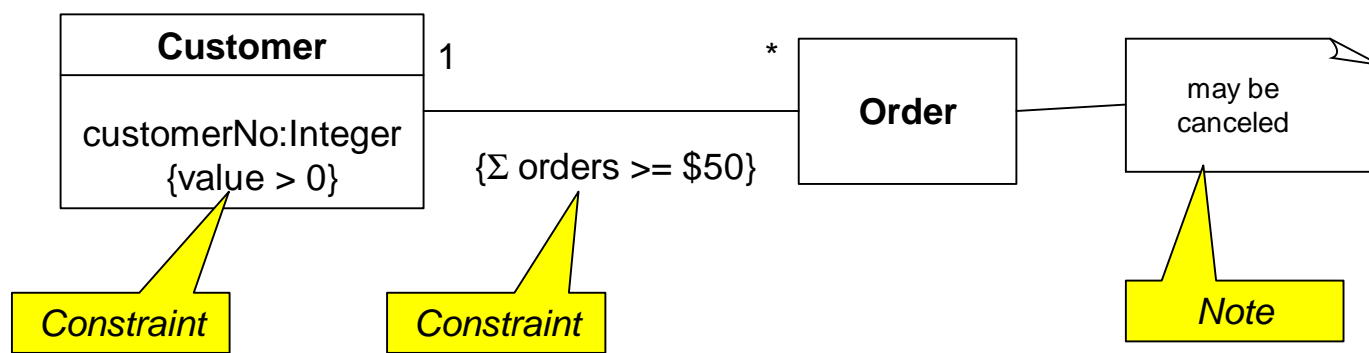


Class Diagrams in Analysis

3.2.10

## Constraint and Note

**Constraints** and **notes** annotate among other things associations, attributes and classes. Constraints are semantic restrictions noted as an expression.



Notes may contain constraints also.

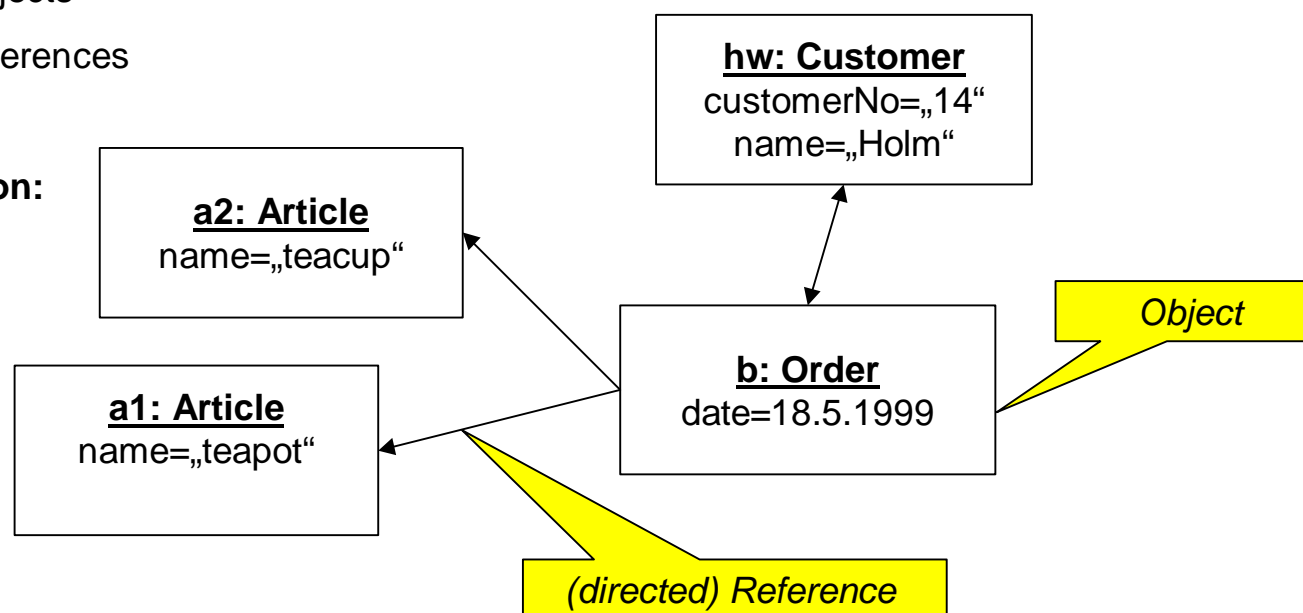
## Object Diagram

An **object diagram** shows objects (instances of classes), optional their states and relationships, at a certain point in time.

**Parts:**

- objects
- references

**Notation:**



## Data Type

---

A **data type** (type) describes a statically defined set of values with **uniform behavior**.

A data type consists of

- ❑ a static **set of values** (a sort)
- ❑ a set of **functions** on values of this set of values (each function with name and signature)
- ❑ the **semantic** of the operations (algebraic or axiomatic)

A type permits the classification of values.

A value may have several types (polymorphism).

You distinguish basic data types and composed data types (data structures).

## Analysis Process

---

**Method** Create conceptual class diagram

**Input:** Specification.UseCaseModel, Specification.Glossary

**Output:** Specification.ConceptualClassDiagram

**Change:** Specification

Specification.ConceptualClassDiagram:= ∅

**repeat**

    Find new **classes** and attributes

    Find new **associations**

    Document classes and relations

    Discuss the result with the customer

**until** customer and developer satisfied

## Classification of the Analysis Process

---

There are several **development processes**

- focussing on the information (data) of the system
- focussing on the functionality (behavior) of the system
- synthesis of both processes

The development process treated in this lecture, described in *Unified Software Development Process* [Jacobson et.al. 1999] for object-oriented analysis is a **synthesis**:

- Initialization by the functionality (use case diagrams)
- Refinement by the information (class diagrams)
- Consolidation by the functionality (collaboration diagrams)

## Analysis: Find Classes (1)

---

### 1. Method: Identify classes by nouns

- Relevant documents: specification, use case documentation, glossary, forms, technical system descriptions, interview notes, ...
- Identify **nouns** (concepts) *being relevant for the problem grasp*
  - Technique: Underline nouns in the text, better too many nouns than too few
- Identify **synonyms**, **acronyms** and special terms.
  - Two different names are synonyms if they designate the same concept.
  - An acronym is a name built out of the initials of a name.
- Identify **homonyms**.
  - A homonym is a *name* describing two (different) concepts.

**Note:** The class diagram details the part of the glossary that is relevant for the specification.

## Analysis: Find Classes (2)

---

### 2. Categorize the nouns

- Concrete objects resp. things, e.g., car
- Persons and their roles, e.g., customer, employee, lecturer
- Information about actions, e.g., bank transfer
- Places, e.g., waiting room
- Organizations, e.g., branch
- Interfaces of the system, e.g., list, choice list
- Relations between classes, e.g., rental agreement between customer and rented object
- General and subject oriented information, e.g., article, seminar type

## Analysis: Find Classes (3)

---

### 3. Delete nouns and concepts, which don't represent any independent conceptual classes

- It's **not** the goal to identify as many classes as possible or simple classes only.
- Deleted nouns may be collected in a **To-Do-List** because they may be helpful to identify relevant attributes, operations and relations.
- Delete names for **roles** which relate a class to another class.
- Indications** for nouns to delete:
  - Neither attributes nor operations can be identified.
  - The noun models implementation details.
  - The class contains only operations which can be attached to other classes.

## Analysis: Find Classes (4)

---

### 4. Choose meaningful and substantial class names

- Each class name should
  - be a noun in singular,
  - be chosen as concrete as possible,
  - be no homonym,
  - be rarely an acronym,
  - represent the entirety of the attributes and operations.

### 5. Document each class in short

- One defining sentence, e.g.,  
*A seminar entry* is a document representing the registration of a customer to a seminar.
- List of the synonyms, optional: distinction from other classes  
**Synonyms:** registration, reservation

## Analysis: Find Attributes

---

### 1. Identify attributes following the noun method

- Identify relevant attributes of each class for the specification.
- Record key attributes only when they are application-specific, e.g., customer number.

### 2. Check the attribute name

- Each attribute name should be
  - a noun in singular or plural,
  - chosen as concrete as possible,
  - no homonym.

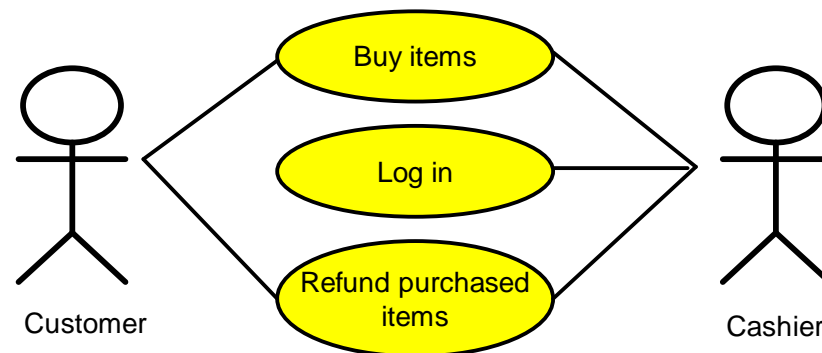
### 3. Define the type of the attributes

- In the conceptual class diagram the type can stay unspecified or specified lazily.
- Complex structured types should be replaced by independent classes.

## Example: *Point of Sale Terminal* (1)

A *Point of Sale Terminal* (POST) is a computer-assisted system to support sales, payments and payoffs in a commercial transaction.

It contains hardware components (computer, display, bar code scanner, ...) and software components.



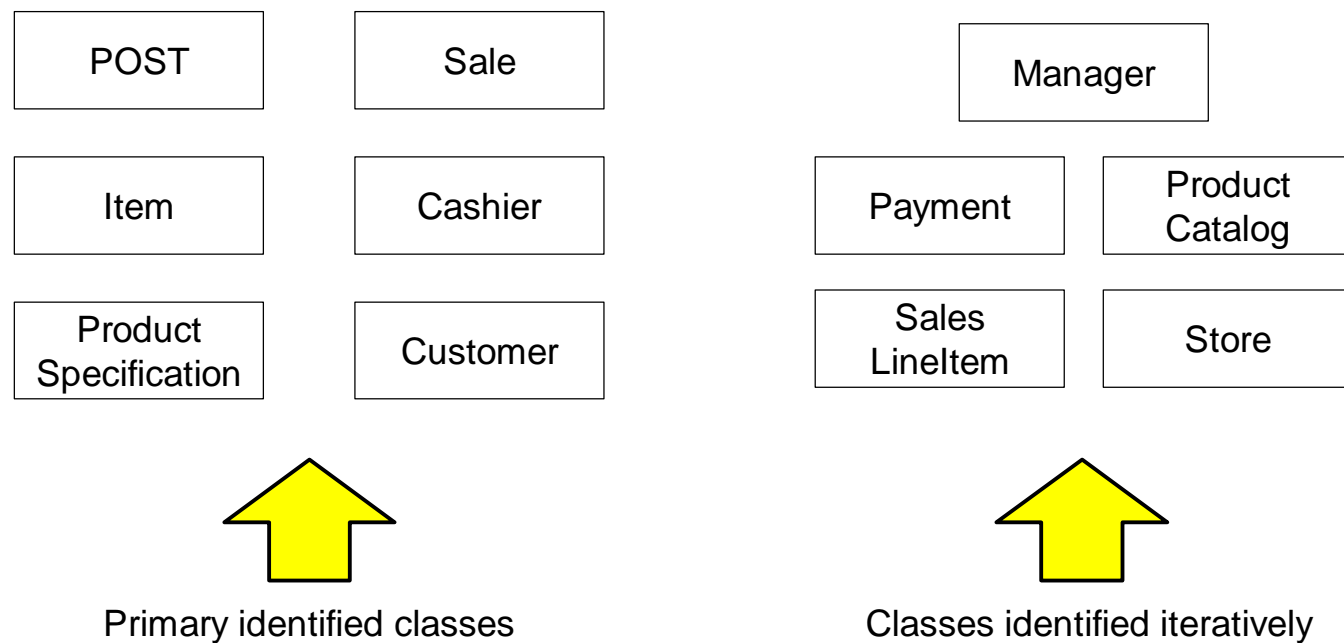
## Example: *Point of Sale Terminal* (2)

Use case description

### Use Case: Buy item

1. This use case begins when a **customer** arrives at a **POST checkout** with **items** to purchase.
2. The **cashier** records the **universal product code** (UPC) from each **item**.
3. The System determines the **item price** and adds the item to the running **sales transaction**.
4. If there is more than one of the same **item**, the **cashier** can enter the **quantity** as well.
5. The **description** and the **price** of the current item are displayed.
6. ...

## Example: *Point of Sale Terminal* (3)



## Analysis: Find Associations (1)

### 1. Identify possible associations between objects

- Find in the relevant documents **verbs** and **nouns** identifying actions or processes in the problem description.
- Identify the concerned classes for each association.
- Prefer binary associations (i.e., with two concerned classes).

### 2. Categorize these associations

- spacial distance (is close to)
- action (drives, books)
- communication (talks with)
- property (has)
- general relation (depends on, is married with)

## Analysis: Find Associations (2)

---

### 3. Delete non-conceptual associations

- Delete non-permanent relations
- Delete associations that are irrelevant for the specification
- Delete implementation-oriented associations

### 4. Define association and role names if necessary

- In most cases associations are defined unambiguously by the participating classes. In this case association and role names are not needed.
- Otherwise (e.g., for recursive associations) define
  - a **name** (a verb or a verbal phrase) for the association
  - a **role name** for each class taking part in the association
  - a **sentence** describing the semantics of the association.

## Analysis: Find Associations (3)

---

### 5. Determine the multiplicity of each role of each association

- Is it a must-relation? ⇒ Cardinality 1..
  - Once the object is created, the relation to the other object must be synthesized, too.
- Is it a can-relation? ⇒ Cardinality 0..
  - The relation can be created at any point in time after the creation of the object.
- Is the ceiling firm or variable? ⇒ Cardinality ..k
  - Is a ceiling **mandatory** regarding the application?
  - In case of doubt always work with the **variable** ceiling!
- Do special conditions apply?
  - Even number, lower limit, ...

**Note:** In case of a firm multiplicity  $k$  (e.g., 2 as lower limit and ceiling) you should consider  $k$  separate associations, if appropriate.

## Analysis: Find Associations (4)

---

### 5. Distinguish associations and aggregations by the following rules of thumb:

- May the relation described by „consists of“ or „is part of“?  
(Collection, container, whole & parts, group & members, ...)
- Is the multiplicity on one side of the association 1 or 0..1?
- Is the association transitive and asymmetric?
- Do the components belong to a subsystem?
- Are the part objects accessed exclusively by the aggregate object?
- Is the lifetime of the component restricted by the lifetime of the aggregate?

#### Note:

**Attributes** are aggregated by their classes, but they don't have any independent existence and so they can't take part in relations independent from their classes.