A special case of an association is the aggregation. Aggregation means an asymmetrical relationship between non-equal partners, e.g., not a peer-to-peer, but a master-slave-relationship.

In aggregation the „whole“ can exist without the „parts“

In composition the „whole“ exists only when the single „parts“ exist. (ex: computer components of a computer)

Notation:
Aggregation and composition (2)

example:

Aggregation

Composition

Car

Wheel

Engine

Frame

4..5

0..1

0..1

1

1

1
Aggregation is a special type of association. The interlinked classes do not have the same status, but a „part of the whole“ relation.

An aggregation describes, how a class, that plays the role of the integrated whole, is composed (aggregated) of other classes that have the role of a part.

Aggregations are represented in UML as associations that end on one side with an empty rhombus.
Compositions are associations with a **very strong** aggregation. The composition is also based on a part of the integrated hole relation, but every relation is so strong, that a part cannot exist independently.

Compositions exist only inside the entity and are destroyed, if the entity is destroyed.

Compositions are represented in UML as associations with a **filled rhombus on one side**.
Inheritance (1)

- An inheritance relationship from class K1 to class K2 is a description of the fact that all objects O of the class K2, in addition to the described characteristics in class K2, have all characteristics of the class K1.

'characteristics' are here

- The list of attributes and participation in associations,
- The list of operations and
- Possibly the associated type and parameter information

Notation:

Colloquial:

Each Class_2 is one Class_1. Class_2 is special from Class_1.
Inheritance (2)

- Helps to model the **similarities and differences between classes**
- **Upper class** (basis class) contains the **shared attributes and operations**
- Each **subclass** (derived) **adds additional individual attributes and operations**.
- A subclass **inherits all attributes, operations, and relationships of the upper class**
- A subclass can **re-write and re-define the implementation of an operation**.
- Each instance of a subclass is at the same time an instance of the upper class
Example:

```
Date
  title: String
  beginning: Date
  duration [0..1]: int
  delayed():void

Personal date
  place: String
  time_to_get_there: int
  authorization(): boolean

Team meeting
  themes [0..*]: String
  room_confirmation(): boolean
  invite(): void
  cancel(): void
```
- The same message leads to activation of different semantically similar methods depending on the receiver (*Dynamic Bond*).
What is defined in K can also represent all objects in classes K, K1 and K2.

What is defined in K is valid only for objects in class K and K2. K1 receives a different meaning (override).

What is defined in K is valid only for objects in class K (and is therefore redundant when K is abstract).
Example of a class diagram: job processing (1)

- Relationship between order and customer
  - an order comes from a single customer
  - A customer can give different orders over time

- Relationship between order and order position
  - Each order has different order positions that refer to a single product

- Relationship between customer and corporate or private customer
  - The class „customer“ has 2 subclasses „corporate customer“ and „private customer“ --> in general

- Relationship between company, customer and employee
  - Multiplicity 0..1 means either there is one or there isn‘t
Example of a class diagram: Job processing(2)

3.2. object oriented modelling with UML

3.2.3. Static Models

- **attributes**
  - Corporate customer:
    - company name
    - credit-worthiness
    - Credit line
    - remind(): void
  - private customer:
    - credit card nr: Integer
  - employee
  - order position:
    - amount: Integer
    - price: money
    - isdelivered: Boolean
  - order:
    - amount: String
    - price: money
    - execute(): void
    - terminate(): void
  - product:
    - amount: Integer
    - price: money
  - customer:
    - name: String
    - address: String
    - credit-worthiness(): void
  - credit-line:
  - execute(): void
  - terminate(): void

- **operations**

- **generalization**

- **associations**

- **multiplicity**

- **multiplicity**

- **class**
Outline

Lecture Content

1. Preface
2. Basics
3. Object orientation
   3.1. Introduction and basic concepts
      3.1.1. Introduction
      3.1.2. Objects and classes
      3.1.3. Attributes of object orientation
   3.2. Object-oriented modelling with UML
      3.2.1. Intro
      3.3.2. Use Case models
      3.2.3. Static models
      3.2.4. Dynamic models
      3.2.5. Summary
   3.3. Methods of the OO Analysis and Design
## Diagram types and application areas

<table>
<thead>
<tr>
<th>Diagram type</th>
<th>Diagram</th>
<th>Phase</th>
<th>Application area</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Requirements</td>
<td>Business processes, general applications</td>
</tr>
<tr>
<td>Use cases</td>
<td>Use case diagrams</td>
<td>Definition, Creation, Delivery</td>
<td></td>
</tr>
<tr>
<td>Static Models</td>
<td>Class diagrams</td>
<td>Definition, Creation</td>
<td>Everywhere, the class diagram is the most Important diagram in UML</td>
</tr>
<tr>
<td></td>
<td>Instance diagrams</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dynamic Models</td>
<td>Activity diagrams</td>
<td>Requirements</td>
<td>Shows the information flow and the time flow of the cooperation of the objects</td>
</tr>
<tr>
<td></td>
<td>Interation diagrams</td>
<td>Definition, Creation, Delivery</td>
<td>Timeflow structure with few messages</td>
</tr>
<tr>
<td></td>
<td>Collaboration diagrams</td>
<td>Requirements</td>
<td>Timeflow structure with few classes</td>
</tr>
<tr>
<td></td>
<td>Sequence diagrams</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>State diagrams</td>
<td>Requirements</td>
<td>Representation of the dynamic behavior</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Definition, Creation, Delivery</td>
<td></td>
</tr>
</tbody>
</table>
Dynamic models

- **Dynamic Modeling** investigates:
  - When does the system do what?
    --> event, reaction
  - What does the system do?
    --> actions, activities
  - Which time-dependent behavior does the system show?
    --> conditions

- The **description tools of dynamic models** are:
  - State diagrams
  - Activity diagrams
  - Interaction diagrams *(Sequence or Collaboration Diagrams)*
### Diagram types and application areas

<table>
<thead>
<tr>
<th>Diagram type</th>
<th>Diagram</th>
<th>Phase</th>
<th>Application area</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use cases</td>
<td>Use case diagrams</td>
<td>• Requirements&lt;br&gt;• Definition&lt;br&gt;• Creation&lt;br&gt;• Delivery</td>
<td>Business processes, general applications</td>
</tr>
<tr>
<td>Static Models</td>
<td>Class diagrams&lt;br&gt;Instance diagrams</td>
<td>• Definition&lt;br&gt;• Creation</td>
<td>Everywhere, the class diagram is the most Important diagram in UML</td>
</tr>
<tr>
<td>Dynamic Models</td>
<td>Activity diagrams&lt;br&gt;Interaction diagrams&lt;br&gt;Collaboration diagrams&lt;br&gt;Sequence-diagrams</td>
<td>• Requirements&lt;br&gt;• Definition&lt;br&gt;• Creation&lt;br&gt;• Delivery</td>
<td>Shows the information flow and the time flow of the cooperation of the objects&lt;br&gt;Timeflow structure with few messages&lt;br&gt;Timeflow structure with few classes</td>
</tr>
<tr>
<td></td>
<td>State diagrams</td>
<td>• Requirements&lt;br&gt;• Definition&lt;br&gt;• Creation&lt;br&gt;• Delivery</td>
<td>Representation of the dynamic behavior</td>
</tr>
</tbody>
</table>
State models

- A state is a characteristic of an object which exists for a limited time. (The characteristics defined in the static modeling are valid for the entire life cycle of the object)
- An event is an occurrence of a negligible duration which has influence on the system in view. An event is described through its name (e.g. the event class) and possibly other parameters
- Examples for events:
  - Creating or deleting an object
  - Sending a message to an object
  - Achieving or termination of a duration (time-out)
- State models describe the change of state of an object by influence of different events
State diagrams

- An object can take on variegated states in its lifetime. With the help of a state diagram, you can see these states, as well as functions, that lead to the changes in the state of an object.

- A state diagram describes a hypothetical machine that at any point in time is in different states. It is comprised of:
  - A beginning state
  - Numerous possible final states
  - Numerous possible end results
  - Numerous number of transitions that describe the transition of the object to the next state
The states are visualized in rounded rectangles bound with arrows. Arrows show the transitions. **Starting condition** is a filled in circle, the **end condition** is an empty circle with a smaller filled circle in the middle.
Example of a State Diagram

1. Start
2. Elevator ready
   - Required floor chosen
3. Elevator active
   - entry: Show required floor
   - do: evaluate required floor
   - do: determine direction of ride
   - do: wait 10 seconds
4. Prepare ride
   - do: close doors
5. Elevator rides
   - do: ride one floor further
   - do: evaluate current floor
6. Prepare to exit
   - entry: show current floor
   - do: open doors
7. Verify ride goal
   - do: evaluate required floor
8. End

Verification:
- Required floor reached
- Other floor chosen
- Start braking
- Current Load<Max
## Diagram types and application areas

<table>
<thead>
<tr>
<th>Diagram type</th>
<th>Diagram</th>
<th>Phase</th>
<th>Application area</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Use cases</strong></td>
<td>Use case diagrams</td>
<td>• Requirements • Definition • Creation • Delivery</td>
<td>Business processes, general applications</td>
</tr>
<tr>
<td><strong>Static Models</strong></td>
<td>Class diagrams</td>
<td>• Definition • Creation</td>
<td>Everywhere, the class diagram is the most Important diagram in UML</td>
</tr>
<tr>
<td></td>
<td>Instance diagrams</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Dynamic Models</strong></td>
<td>Activity diagrams</td>
<td>• Requirements • Definition • Creation • Delivery</td>
<td>Shows the information flow and the time flow of the cooperation of the objects</td>
</tr>
<tr>
<td></td>
<td>Interaction diagrams</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Collaboration diagrams</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Sequence-diagrams</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>State diagrams</td>
<td>• Requirements • Definition • Creation • Delivery</td>
<td>Representation of the dynamic behavior</td>
</tr>
</tbody>
</table>
Activity diagrams

- **Activity diagrams** describe the action possibilities that are comprised out of single activities (steps)
- The course of a use case is frequently described by an activity diagram - it is suited for the modeling of all activities within a system
- The **activity diagram** is comprised of:
  - **Activities**: modeling elements, that represent the execution of a set of operations
  - **Transitions**: shows the transition from one activity to the next. Can be tied to other conditions
Activity diagrams and their graphical notation

- **Beginning condition**
- **Transition to new activity**
- **End condition**

**Object condition:** Link of activities with conditions

**Decision:** select a transition with condition1

**Synchronization of parallel activities**
Activity diagrams and their graphical notation

Lane markers:
Simplify the responsibility
Activity Diagrams and their Graphic Notation

• Activity diagram:
  - An activity is shown by a rectangle with rounded edges and contains the description of an internal action. From the activity, transitions start, representing the end of the internal action and the transition to the next activity. The transition is shown by empty peak arrows, pointing towards the new activity. Between two activities, an optional reference on the state of the object can be shown by a dashed open arrow.