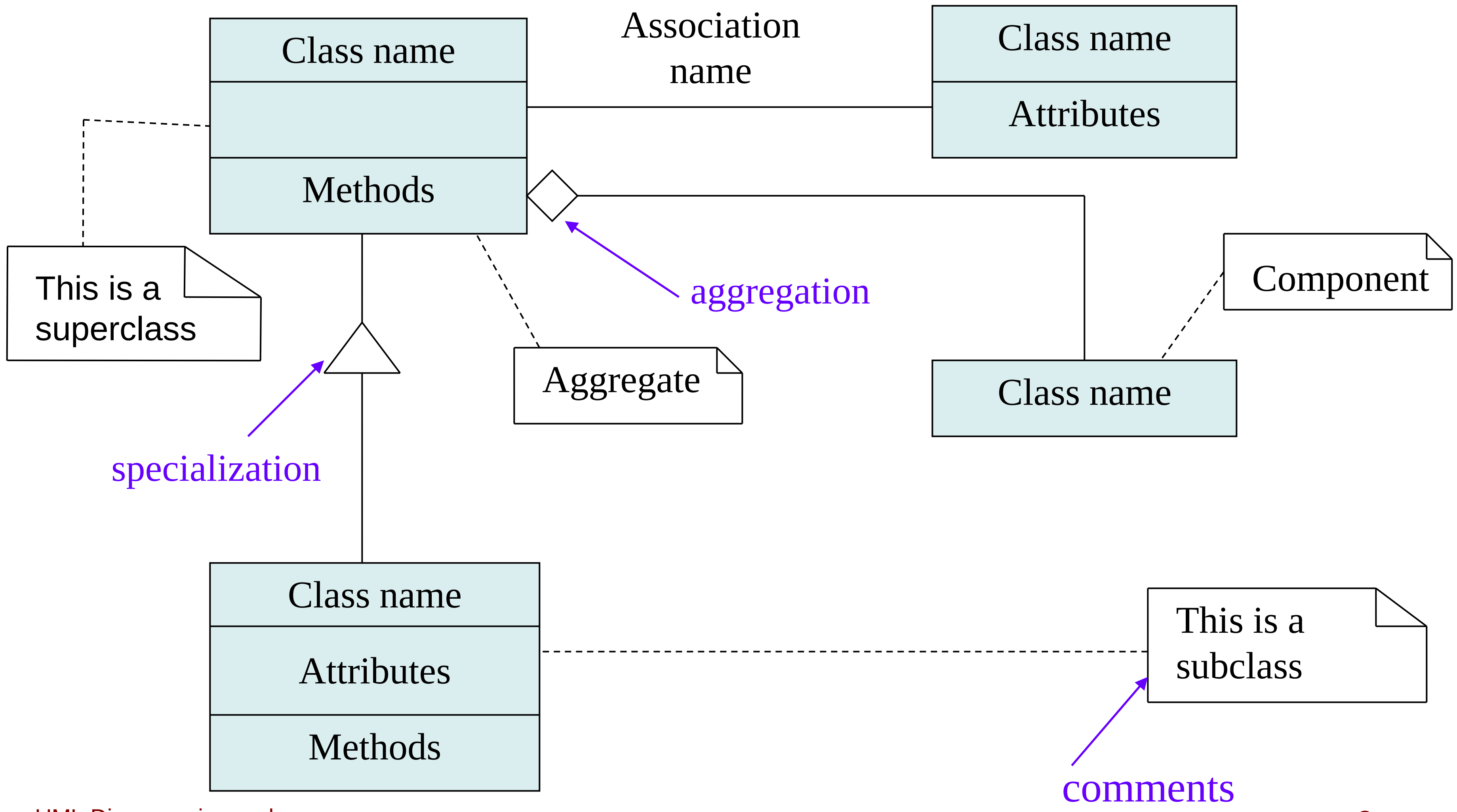
# Class Diagram

Alif Finandhita

Teknik Informatika – Universitas Komputer Indonesia

## Class and Object Diagram Modelling

# Class diagram – basic syntax



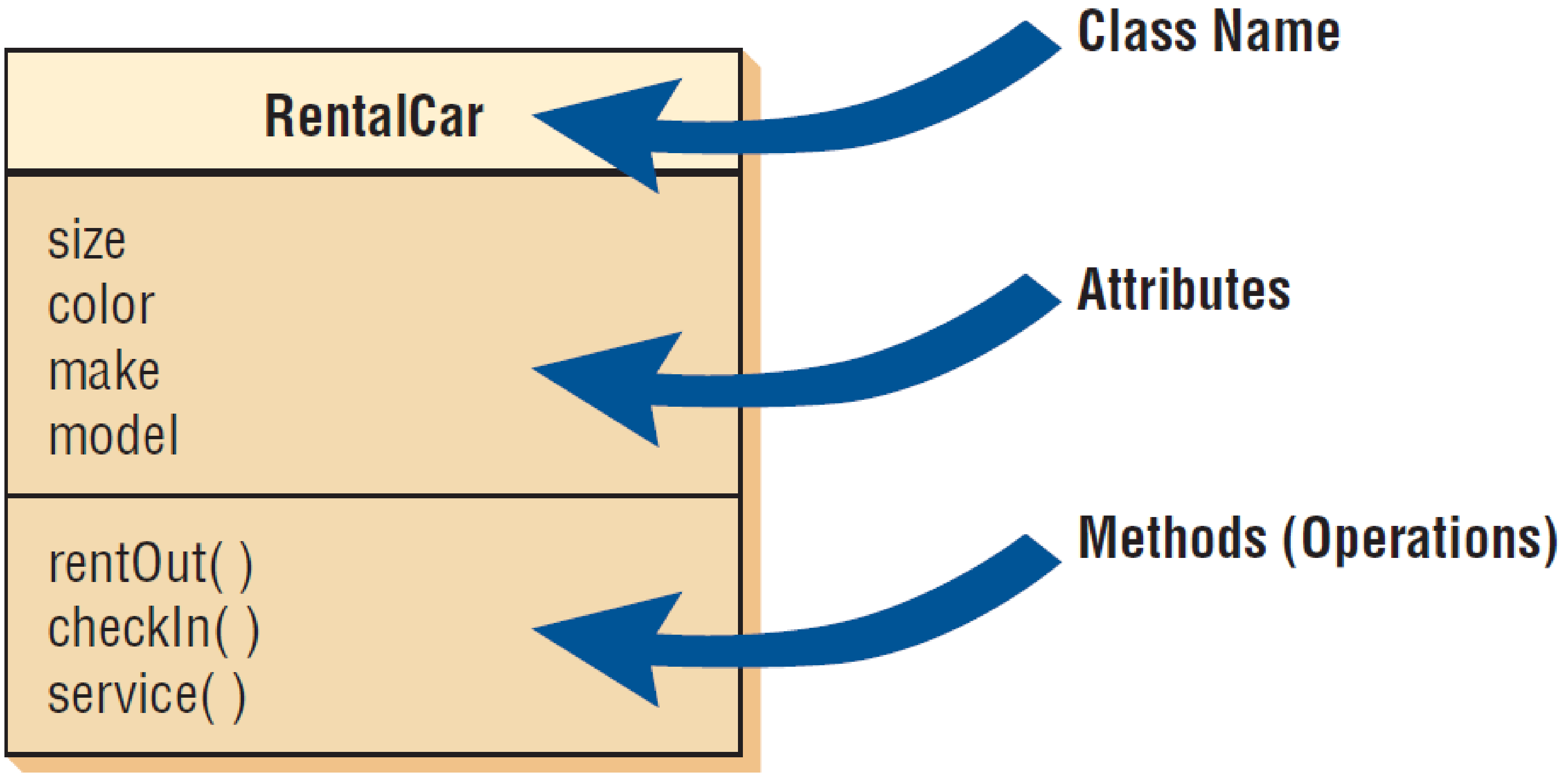
## Class Diagram: semantics (1)

- A class icon must have a class name. Optionally, it can have attributes and/or methods.
- Attributes and methods are strings and will not be validated by the modeling tools.
- Attributes can be specified by their names or by <name : type> pairs.
- Methods can be specified by their names or with partial signatures or with complete signatures.

## Class Diagram: semantics (2)

- Comments can be included in any diagram with a rectangle folded at right top corner
  - The dotted line from the comment is important to indicate which portion of the diagram is explained by the comment
- Suggestion
  - For validation purposes, when showing aggregation relationship, the aggregate (the one near the diamond edge) must include an attribute whose type is the component class

# Class


RentalCar
size
color
make
model
rentOut( )
checkIn( )
service( )
Class Name
Attributes
Methods (Operations)

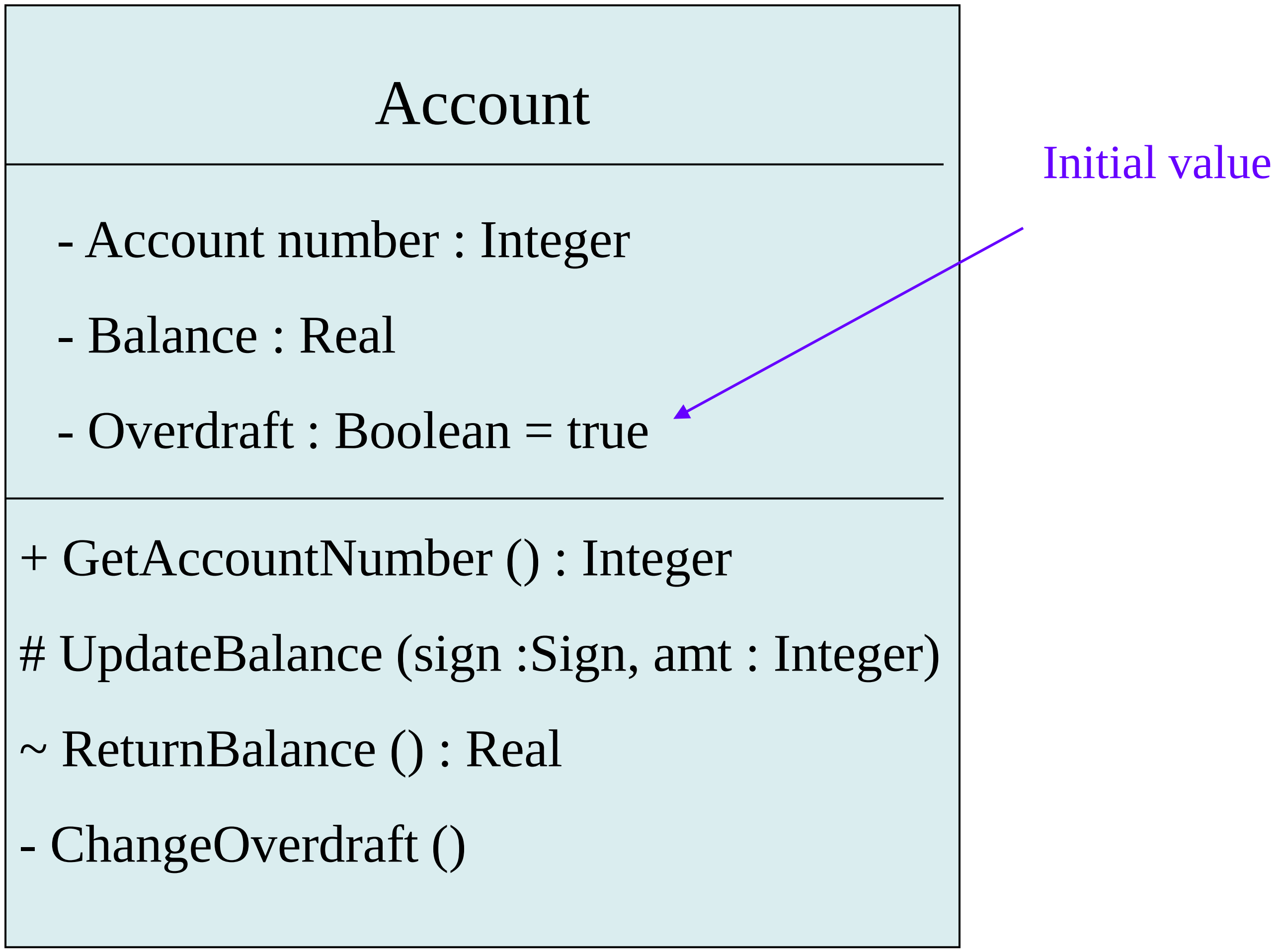
# Details of a class icon

**+ public**

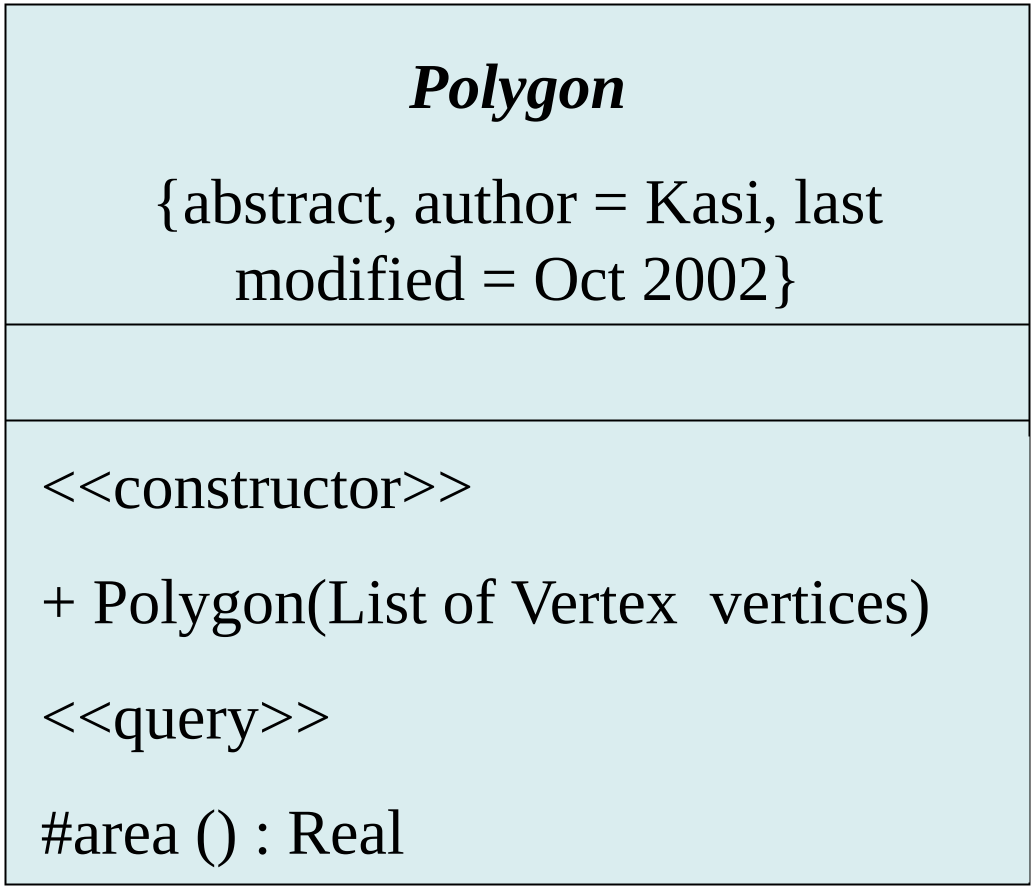
**- private**

**# protected**

**~ visible within the package**


Account
- Account number : Integer
- Balance : Real
- Overdraft : Boolean = true
+ GetAccountNumber () : Integer
# UpdateBalance (sign :Sign, amt : Integer)
~ ReturnBalance () : Real
- ChangeOverdraft ()
Initial value

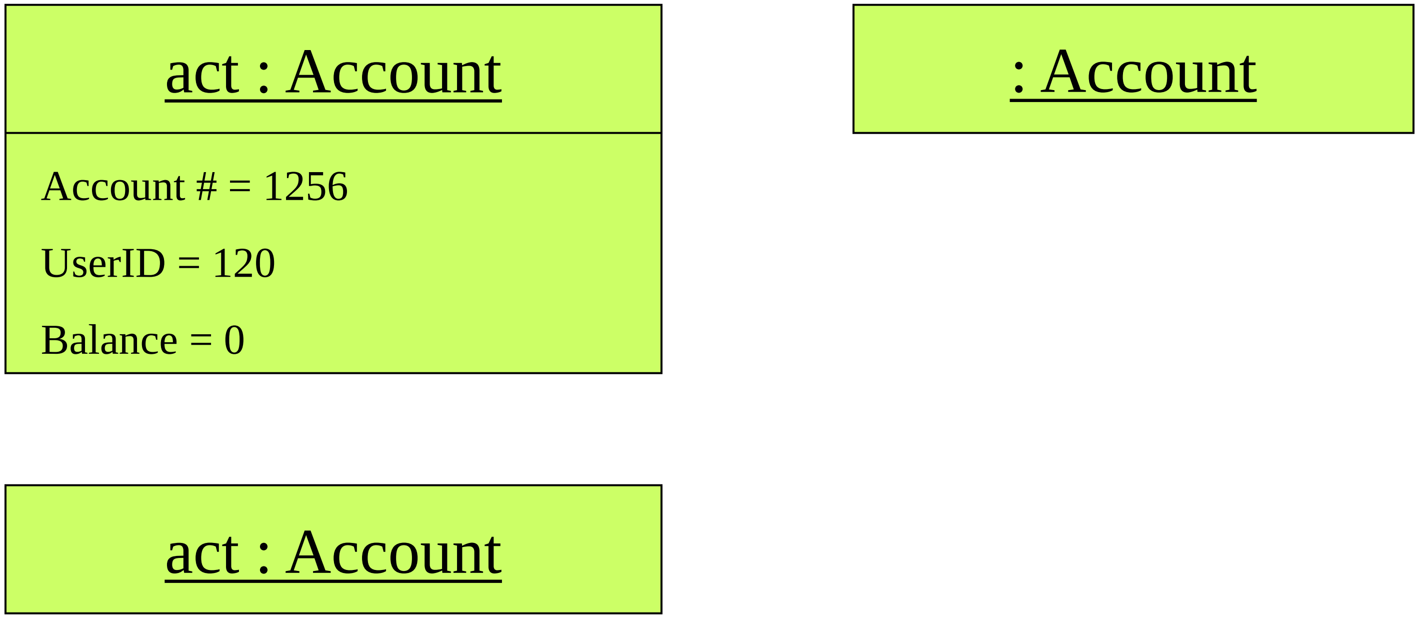
# An abstract class


*Polygon*
{abstract, author = Kasi, last modified = Oct 2002}
<<constructor>>
+ Polygon(List of Vertex  vertices)
<<query>>
#area () : Real

## Multiplicity

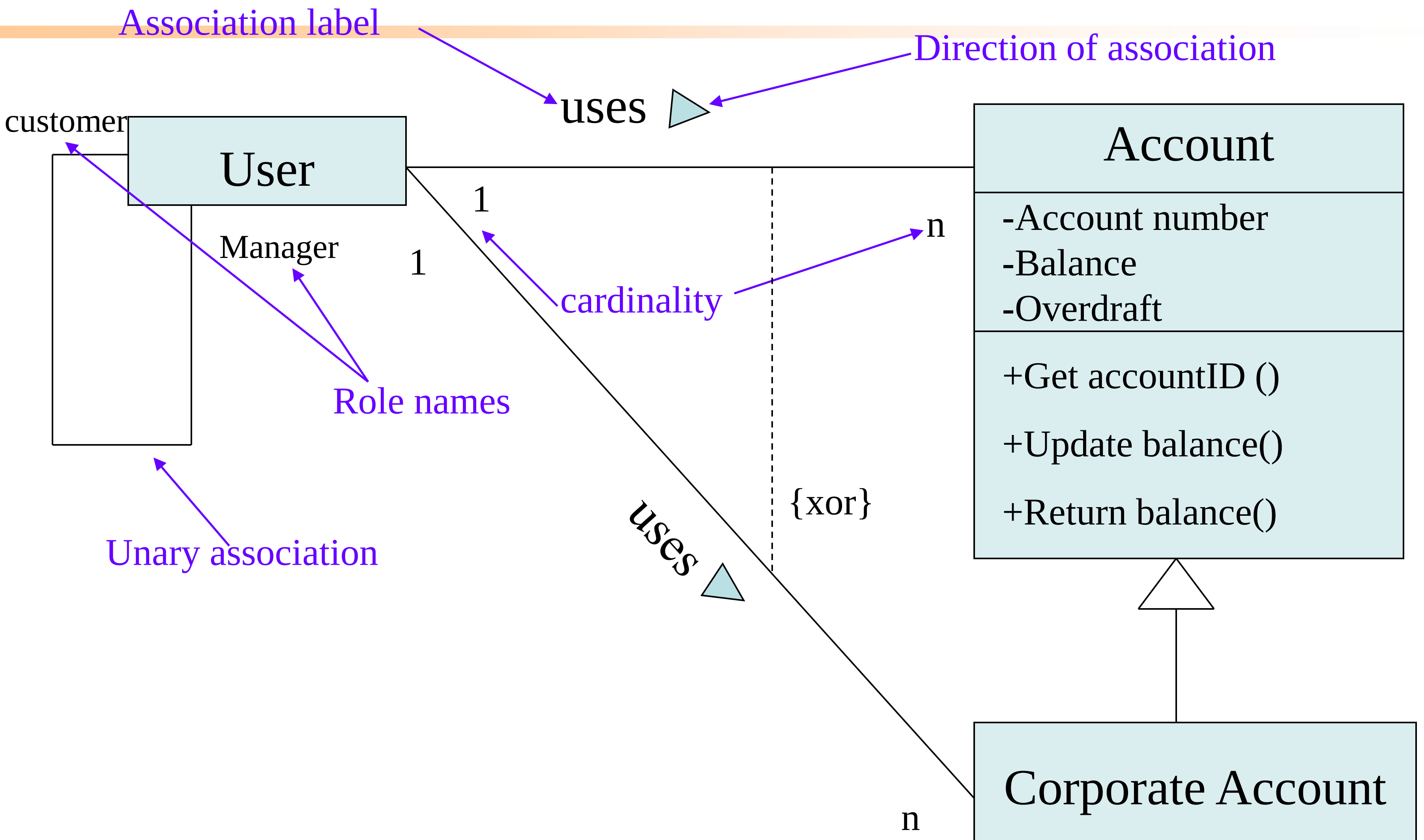
| Multiplicity | Explanation |
|---|---|
| 0..1 | 0 or 1 |
| 1 | Mandatory 1 |
| 0..* | 0 or Many |
| 1..* | 1 or Many |
| * | Many |

# Object Diagram



**Various representations of an account object**

# Association –syntax



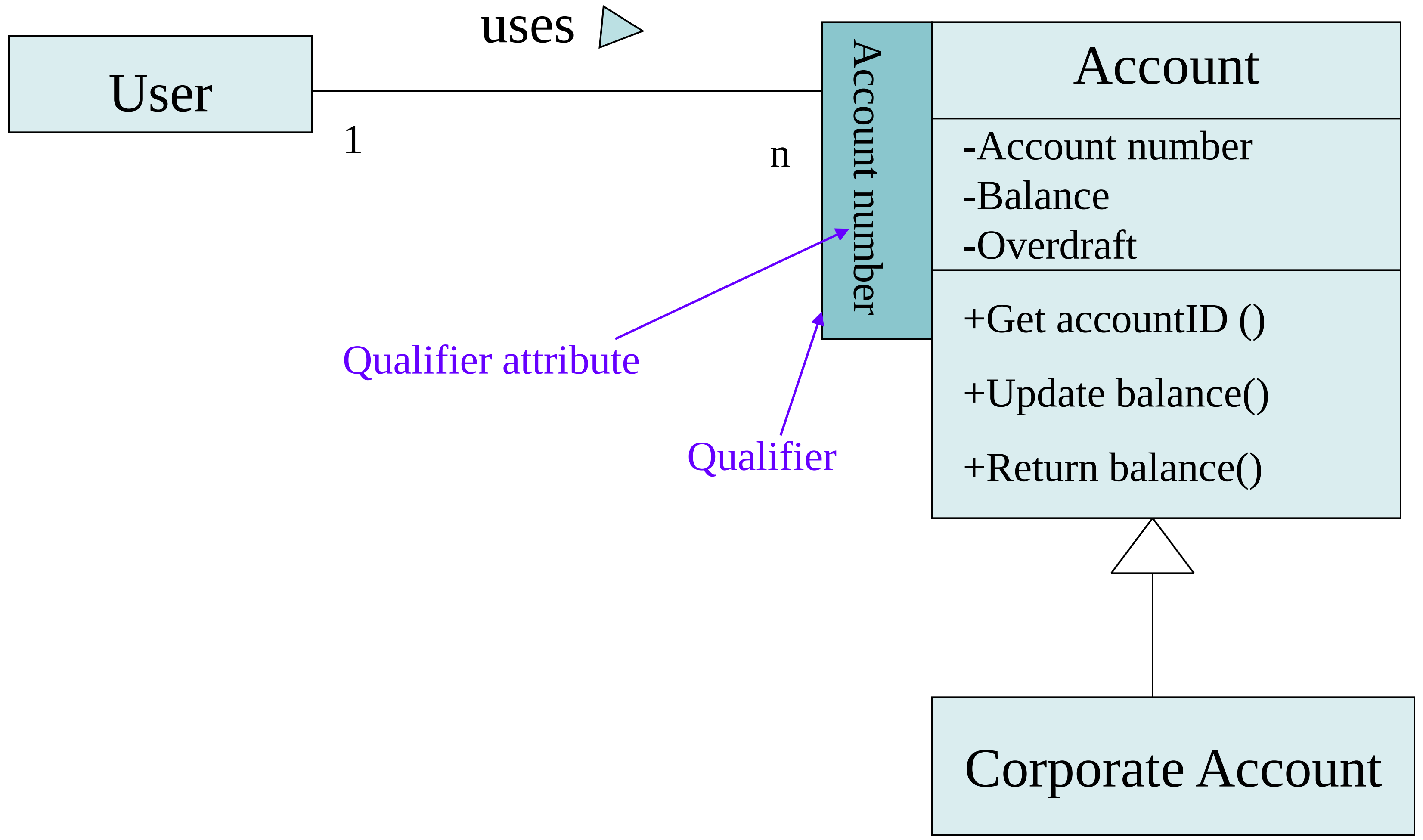
## Association – Semantics (1)

- Every association is expected to be labeled
  - UML does not require a name for an association
- Direction of an association, cardinality, role name are all optional
  - For unary associations, it is better to include role names
- Representations of cardinality
  - 0, 1, * (zero or more), n..m (values in the range between n and m both inclusive)

## Association – Semantics (2)

- A constraint may be [optionally] placed between two associations
  - See the example in the previous slide that asserts an Exclusive OR relationship between the associations
- When a subclass specializes a superclass, it also inherits all associations between the superclass and other classes
- In the previous example, the association “uses” between “User” and “Account” is also inherited by the pair “User” and “Corporate Account”

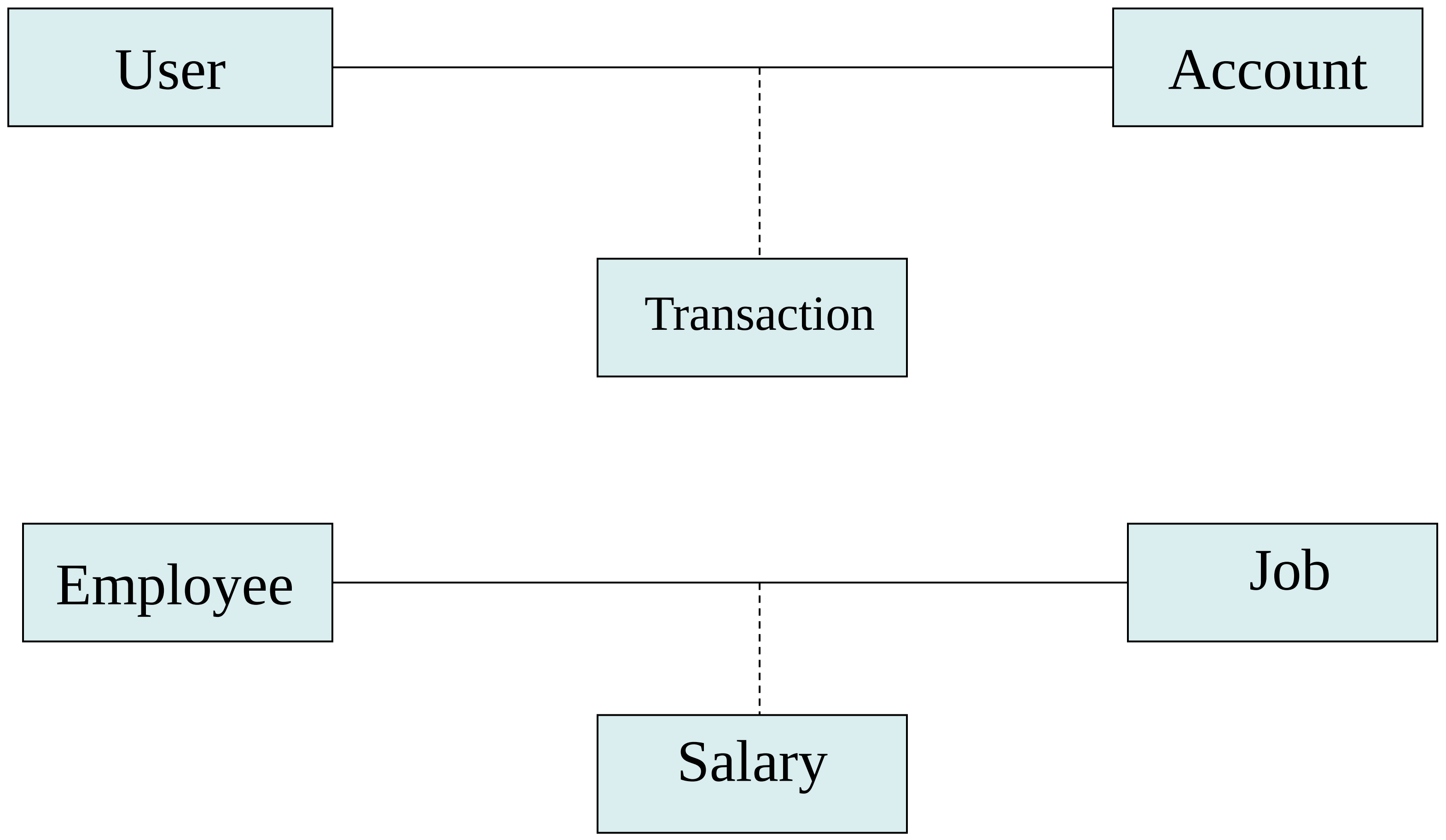
# Association with qualifiers



## Association - Qualifiers

- Qualifiers can be attached to a “one-to-many” association
  - It is rectangle attached to the “many” end of the association
- A qualifier is a collection of variables whose values uniquely identify an instance at the “many” end of the association
  - In the example, an account number uniquely identifies an account in a collection of accounts
- Qualifier is part of the association

# Association Class

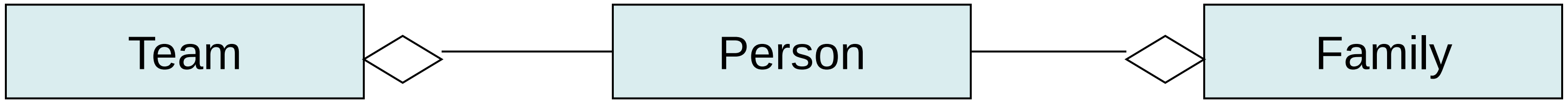


## Association Class - semantics

- A piece of information that belongs to both classes in an association is put into a separate class called “association class”
  - Association class is a dependent class that depends on the other two classes in the association
  - An association class cannot exist independently
  - An object of an association class must refer to objects of the other two classes in the association
    - Example: A “Transaction” object depends on a “User” object and on an “Account” object.

## Shared Aggregation

An aggregation relationship in which the component can be shared by classes/objects outside the aggregation



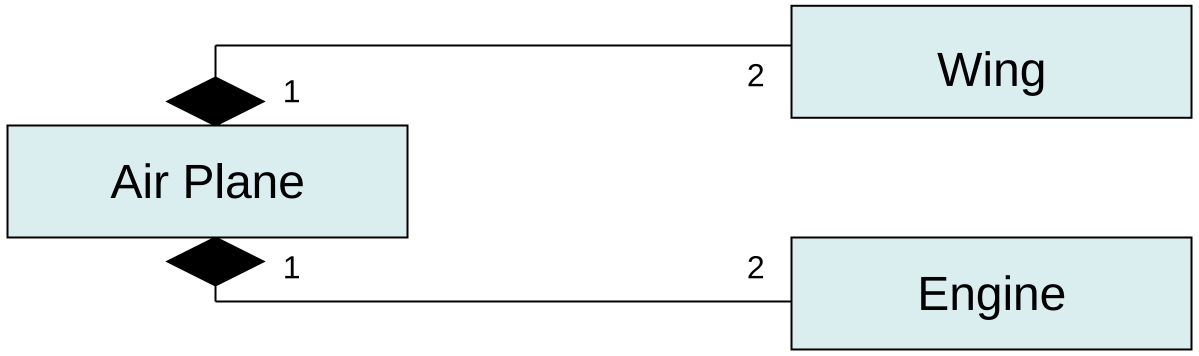
Person object is shared by both Team and Family objects

Shared aggregation is indicated by a hallow diamond

**Caution:** Changes made to a component object will affect all the aggregates that include the component.

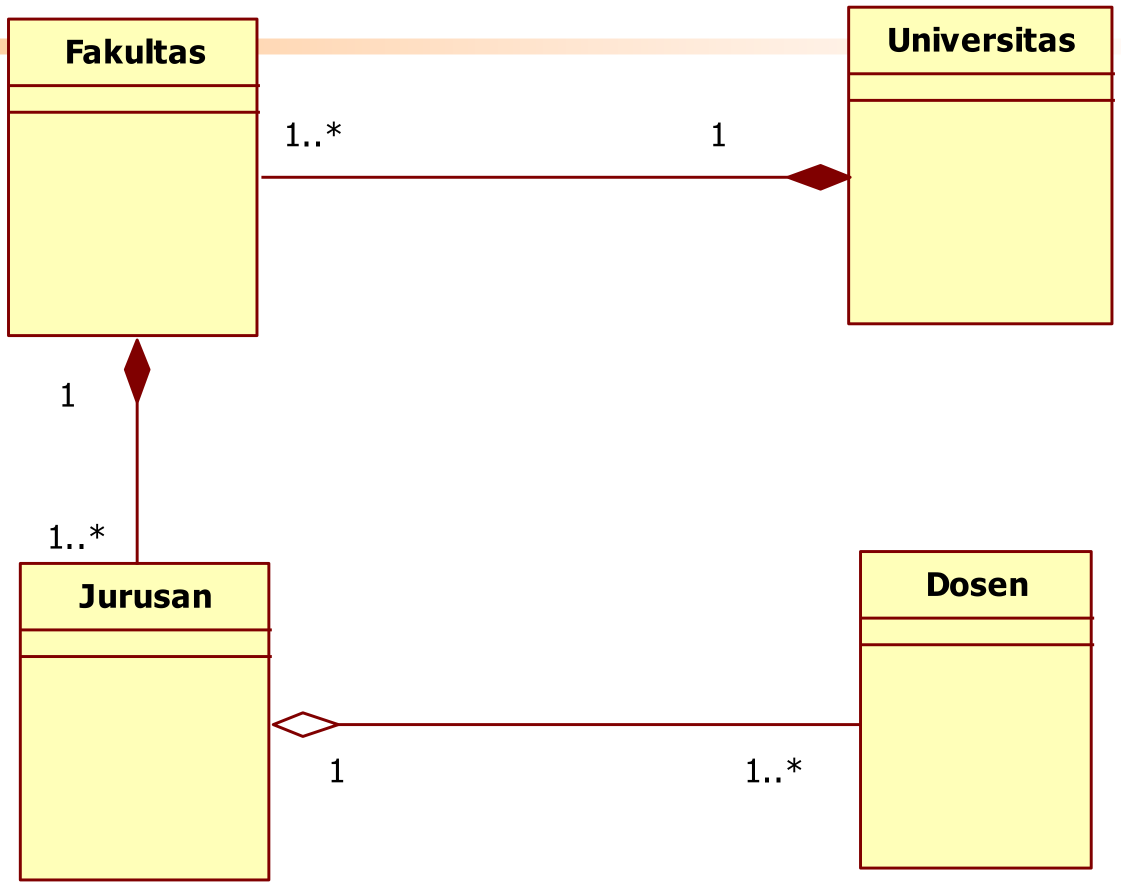
# Composite Aggregation

An aggregation relationship in which the component is an exclusive part of the aggregate; hence, not shared.



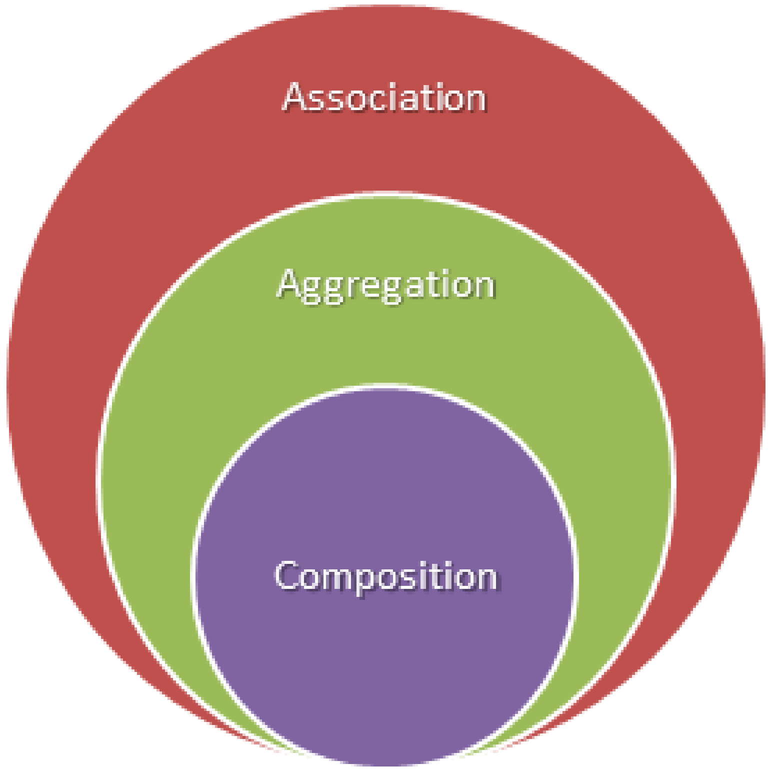
Composite aggregation is indicated by a filled diamond

# Composition VS Aggregation

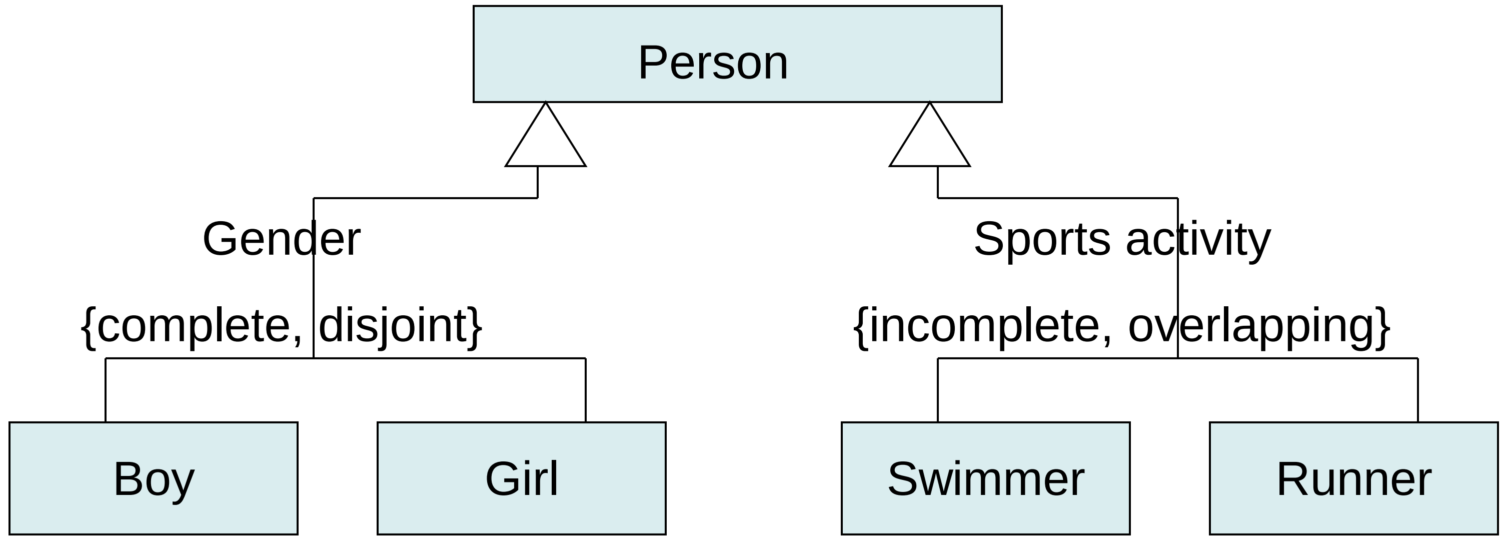


Bila Universitas ditutup maka Fakultas dan Jurusan akan hilang akan tetapi Dosen tetap akan ada. Begitupun relasi antar Fakultas dengan Jurusan

## Association VS Composition VS Aggregation



# Advanced Specialization



These optional domain words make the relationships easier to understand.

## How to find classes? (1)

- Nouns in requirements document or use case descriptions may provide a good starting point, but often are inadequate
- Each class should contain a distinct set of operations relevant to the system under consideration
  - Think of a class as an ADT
- Remove vague classes
  - Classes that do not adequately describe themselves
    - A class that represents the internet

## How to find classes? (2)

- Try not to include implementation-oriented classes in the analysis model
  - May be introduced later during design and/or implementation
  - Examples: array, tree, list
  - These classes will not only occupy so much space in the diagram but also tend to divert the focus of analysis

## How to identify associations? (1)

- An association corresponds to a semantic dependency between classes
  - Class A uses a service from class B (client-server)
  - Class A has a structural component whose type is class B (aggregation)
  - Class A sends data to or receives data from class B (communication)

## How to identify associations? (2)

- Include only those associations that are relevant to the current model
    - Constrained by assumptions, simplifications, system boundary (what is expected to be provided by the system)
    - Three different associations between “Faculty member” and “Course”
        - “Faculty member” teaches “Course” in a course registration system
        - “Faculty member” creates “Course” in a curriculum development system
        - “Faculty member” evaluates “Course” in a course evaluation/inspection system

## How to identify associations? (3)

- Eliminate redundant associations
  - “Faculty member” teaches “Course”
  - “Course” is taught between “Time” to “Time”
  - Therefore, “Faculty member” teaches between “Time” to “Time”
    - use transitivity between associations
- Remember that subclasses inherit the associations of a superclass

## How to identify aggregations?

- Aggregations are also associations
- Identify as Association if it is not clear whether it is Association or Aggregation
    - “Mail” has “Address” (aggregation)
    - “Mail” uses “Address” for delivery (association)
    - “Customer” has “Address” (aggregation)
    - “Customer” resides at “Address” (association)
    - “TV” includes “Screen” (aggregation)
    - “TV” sends information to “Screen” (association)

## How to identify specialization?

- Generally, specialization relationships are noticeable in the application domain
- Top-down approach
  - “Student”, “Full-time Student”, “Part-time Student”
  - “TV”, “Plasma TV”, “Flat Panel TV”
  - “Customer”, “Bank manager”, “Teller”

## How to identify specialization (2)

- Some of them are discovered during analysis
- Bottom-up approach
  - “Part-time Instructor” derived from “Instructor” and “Student” while modeling a department
  - “User” derived from “Customer”, “Bank Manager” and “Teller” while modeling an ATM system
  - “Material” derived from “Book”, “Journal” and “Magazine” while modeling a library catalog system

## References

1. Roger S. Presmann, Software Engineering, 6th edition.
2. Kendall, System Analysis and Design, 7th edition.
3. Ian Sommerville, Software Engineering, 8th Edition
4. PPT of Roger S. Pressman (chung and zheng)
5. PPT of Kendall
6. Saiful Akbar, Handouts PPL – ITB, 2011
7. Scott W. Embler, Elements of UML Style 2.0
8. Martin Fowler, UML Distilled 3, Third Edition