Object-Oriented Programming

Object modelling and relations between objects

Anders Jonsson & Federico Heras
2023-24

Course topics

Topic 1 Introduction and the concept of objects

Topic 2 The object-oriented programming paradigm

Topic 3 Object modelling and relations between objects

Topic 4 Inheritance and polymorphism

Topic 5 Abstract classes and interfaces

Topic 6 Reuse and study of problems solved using objects

Object-oriented design

1. Identify the objects that will participate in the solution
2. If an object is already defined, reuse and/or modify
3. If an object is not defined, create a new definition
4. Specify the relations that exist among objects
5. Identify the type of each relation
6. Determine how the objects interact in the solution

Types of relations

- There exist five fundamental types of relations:
  1. Generalization/specialization or inheritance (is a)
  2. Composition/aggregation (is part of)
  3. Use/dependency (uses)
  4. Association (general relation)
  5. Template/generics

Generalization/specialization

- Represented by the property of inheritance
- Can be expressed by the verb is a
- Transitive relation:
  - A car is a vehicle
  - A vehicle is a means of transport
    $\Rightarrow$ a car is a means of transport
- Bidirectional: A generalizes B if B specializes A

Composition/aggregation

- Represents composite objects
- Can be expressed by the verbs is part of or has a
- Bidirectional:
  - The car has an engine
  - The engine is part of the car

Use/dependency

- "Indirect" relation between A and B
- No attributes that store instances of the other class
- Temporary use:
  - Class A has a method that contains arguments of type B
  - Class A creates instances of class B

Association

- Relation "by default"
- Semantic connection that does not correspond to one of the other types
- Normally bidirectional, but can be unidirectional
- Defines the roles that exist among different objects
- Normally binary, but can be unary or ternary
- Also implies that a class has attributes of another class

Theory session 4

Cardinality

Association classes

Templates

Exercises

Cardinality

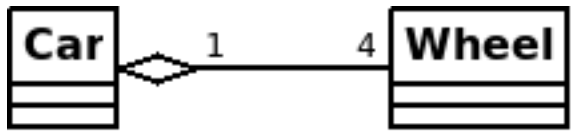
- The number of instances of a class that are related to instances of another class
- Examples:
  - One teacher gives a class to tens of students
  - One students has one academic record

Cardinality

- Suppose there are two related classes A and B
- Card$_{A,B} = X$: each instance of A is related with X instances of B
- The cardinality between A and B is defined in both directions
  - One-to-one: Card$_{B,A} = 1$, Card$_{A,B} = 1$
  - One-to-many: Card$_{B,A} = 1$, Card$_{A,B} = n$
  - Many-to-many: Card$_{B,A} = m$, Card$_{A,B} = n$
  - Zero-to-one (optional): Card$_{B,A} = 0/1$, Card$_{A,B} = 1$

Cardinality

- Important property: defines the attributes of each class




```
public class Car {
    private Wheel w1, w2, w3, w4;
}
```

- Good idea to always specify cardinality in both directions
- Exception: inheritance (cardinality is always one-to-one)

Theory session 4

Cardinality

Association classes

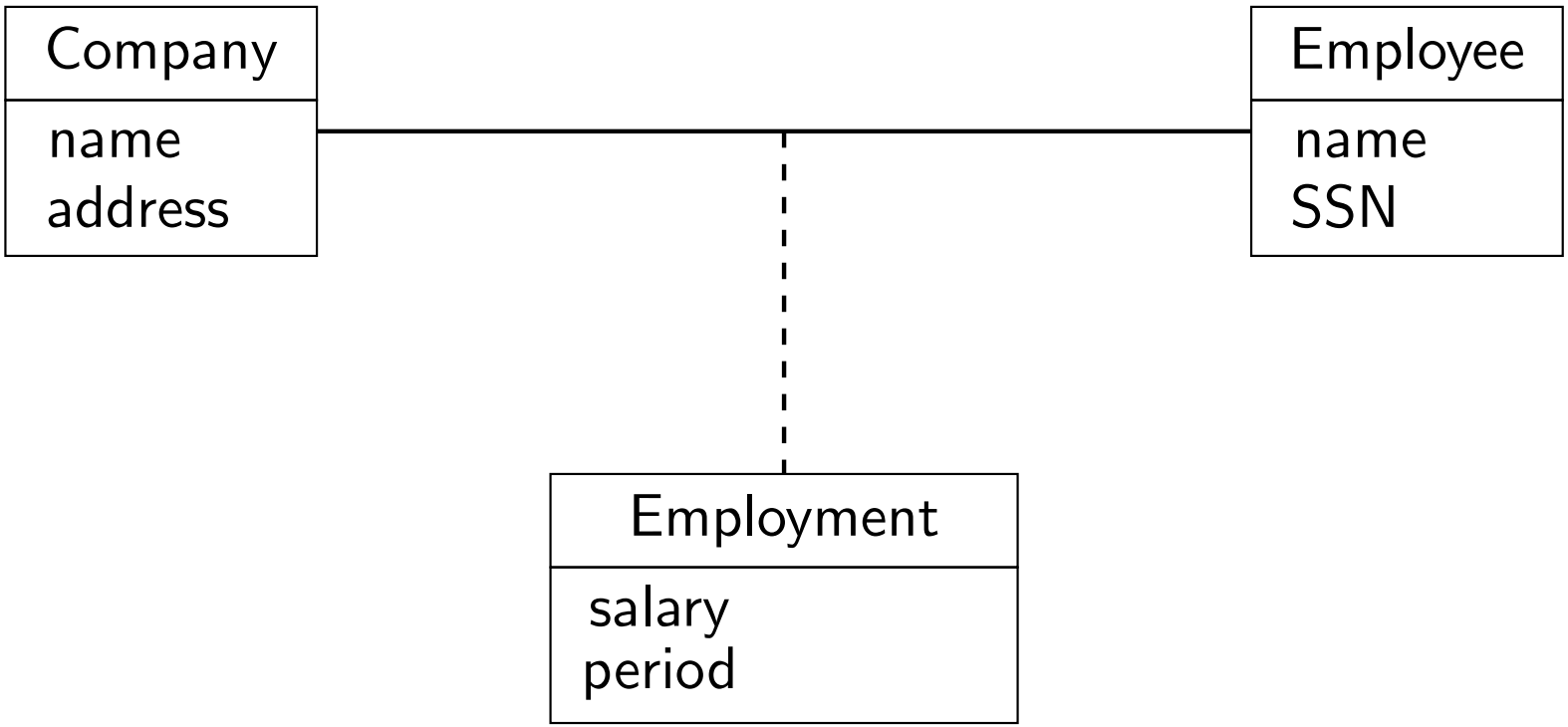
Templates

Exercises

Association class

- Some data can be associated with the relation itself
- This data does not belong in one of the two related classes
- Solution: define an association class that contains the data
- Do not confuse “association class” with “association relation”!
- An association class is not the same as a ternary association

Example association class



Theory session 4

Template/generics

- Relation that makes it possible to create generic definitions
- Not supported in all programming languages
  - C++: includes support for templates
  - Java: includes support for generics
- Syntax that makes it possible to define abstract types that are instantianted at compile time

Template/generics

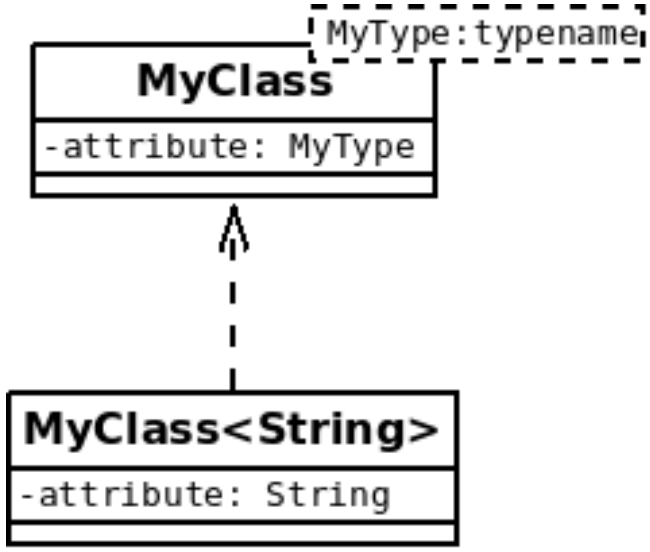
- Generic programming: code that works for many different cases

```
public class MyClass< MyType > {
    private MyType myAttr;
    public MyClass( MyType attr ) { myAttr = attr; }
}
```

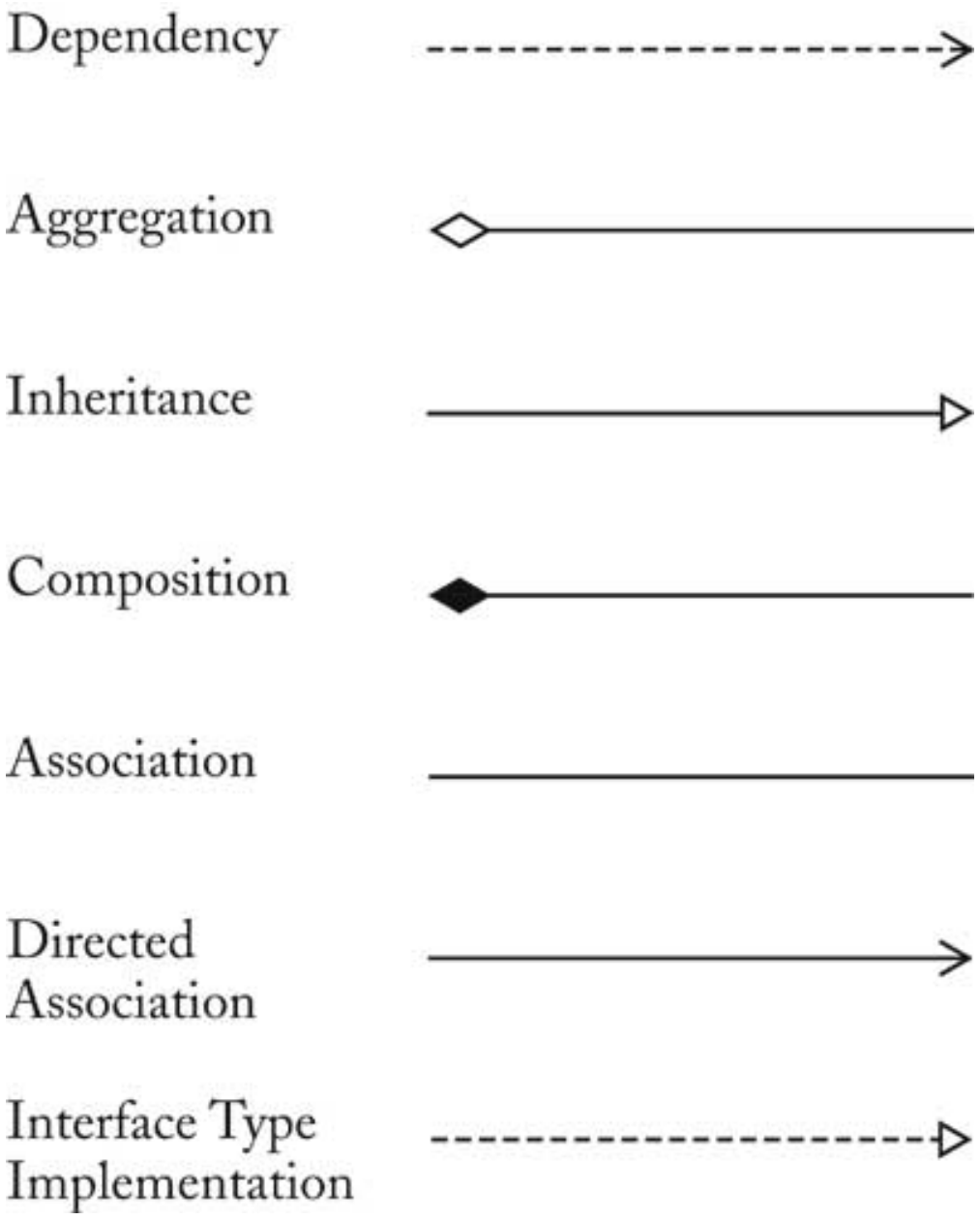
- Restrict the use of generic references to concrete types

```
MyClass< String > myInstance;
myInstance = new MyClass< String >( "Hello" );
```

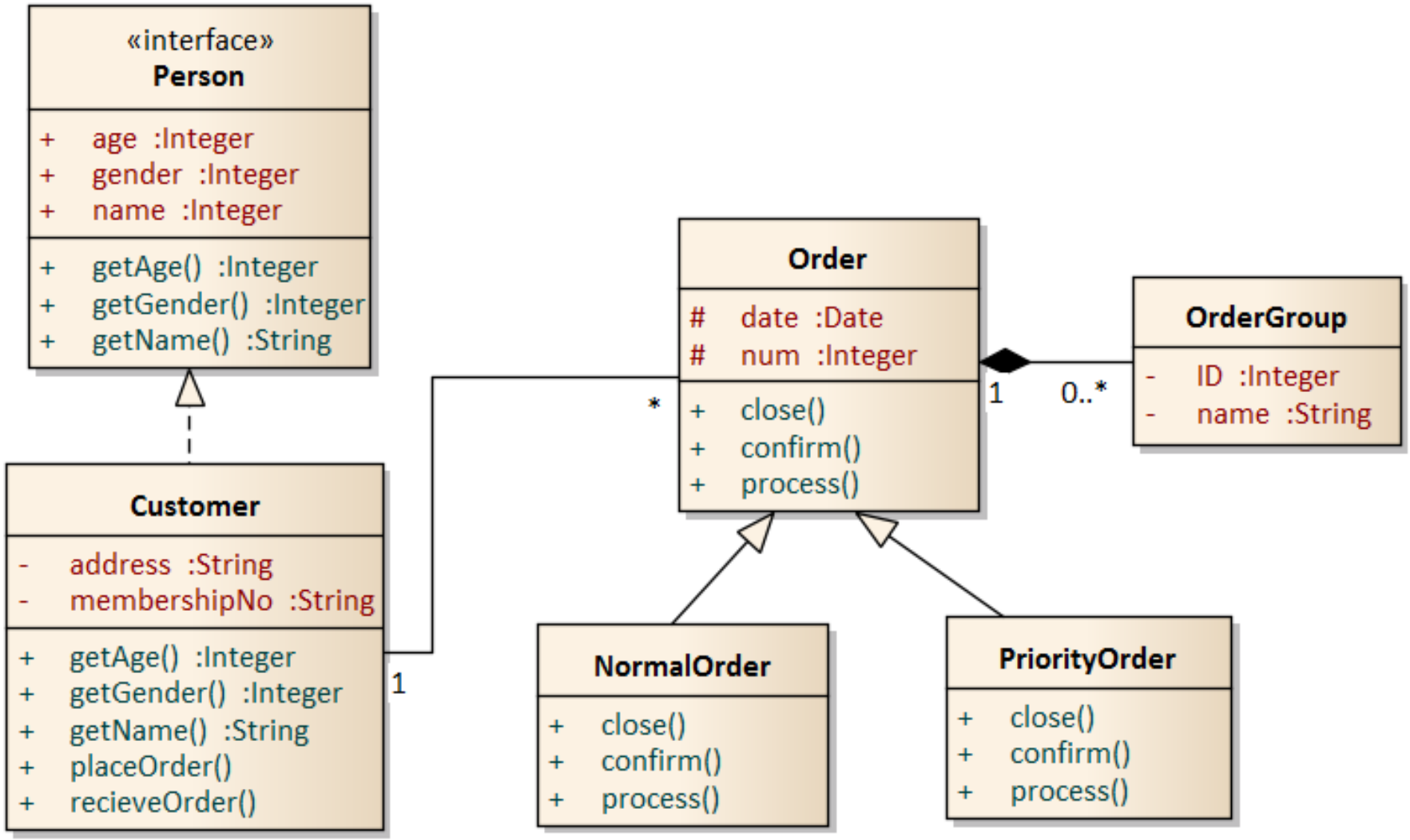
Example template/generics



Graphical representation



Class diagram



Theory session 4

Exercise

- Draw the class diagram for the classes from the first seminar

Exercise

- Design a program that represents a music collection. Include classes for songs, artists, albums and playlists.
- Implement the program in Java

Exercise

- Draw a class diagram that represents a software company with the following characteristics:
  - Employees are grouped in development teams
  - Employees may be part-time (hourly salary) or full-time (monthly salary)
  - In each development team a full-time employee acts as director
- (Optional) Implement the classes in Java

Exercise

- Draw a class diagram that represents the entities of a university:
  - The university is composed of departments that hire professors
  - Each course is linked to a department and each professor can teach one or more courses
  - The professors may be permanent or part-time
  - The students are enrolled in one or more courses
  - The students can be undergraduate or master
- (Optional) Implement the classes in Java

Summary

- Cardinality: number of instances related to each other
- Association class: data induced by a relation
- Templates: generic programming with abstract types