University of Applied Sciences Northwestern Switzerland
School of Business

Master of Science
Business Information Systems

UML - Modelling Data

Knut Hinkelmann

References

- OMG Unified Modelling Language - UML, Current Standard Version 2.1.2 http://www.omg.org/spec/UML/2.1.2/
- R. Miller: Practical UML: A Hands-On Introduction for Developers. http://edn.embarcadero.com/article/31863
- Donald Bell: UML basics: The class diagram. http://www.ibm.com/developerworks/rational/library/content/RationalEdge/sep04/bell/

Unified Modeling Language UML

- Unified Modeling Language (UML) is a set of standardized modeling languages in the field of software engineering.
- UML includes a set of graphic notation techniques (diagrams) to create visual models of software-intensive systems, including their structure and design
- In UML, you can model
  - any type of application,
  - running on any type and combination of hardware, operating system, programming language, and network
- The UML standard is developed and managed by the Object Management Group OMG and forms a foundation of OMG's Model Driven Architecture (MDA)
  - a UML model can be either platform-independent or platform-specific,
- Using XMI (XML Metadata Interchange, another OMG standard), it is possible to transfer a UML model
  - from one tool into a repository, or
  - into another tool for refinement or the next step in your chosen development process.

Source: Introduction to OMG's Unified Modeling Language™ (UML®), http://www.omg.org/gettingstarted/what_is_uml.htm

Types of UML Diagrams

UML contains diagrams for modelling structure (data and IT) and behavior of software systems

Structure diagrams

Data

1. Class diagram
2. Object diagram

IT systems

3. Component diagram
4. Deployment diagram
5. Composite structure diagram (*)
6. Package diagram

Behavior diagrams

7. Use-case diagram
8. State machine diagram
9. Activity diagram

Interaction diagrams

10. Sequence diagram
11. Communication diagram
12. Interaction overview diagram (*)
13. Timing diagram (*)

(*) not existing in UML 1.x, added in UML 2.0

Object Orientation

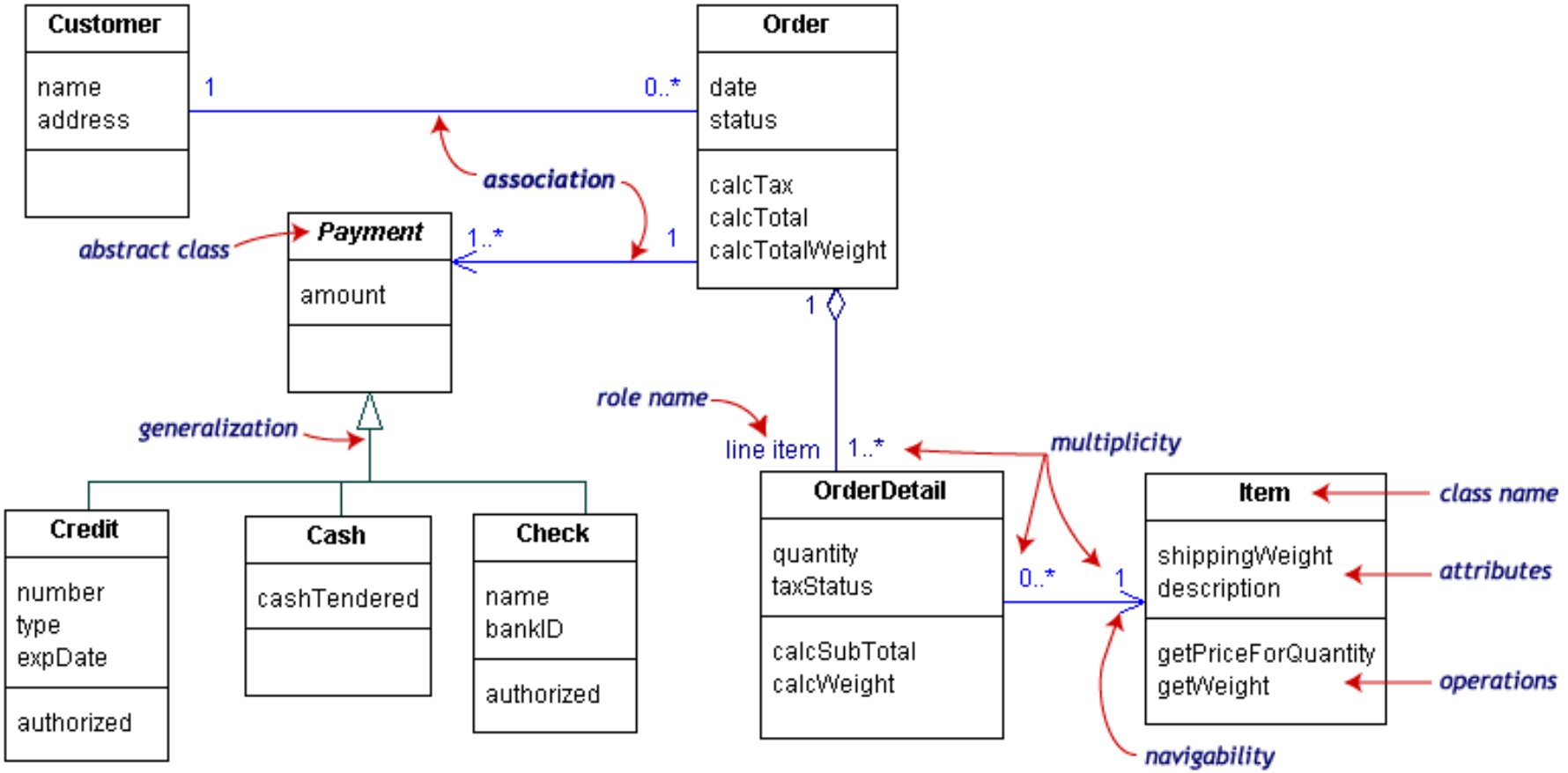
- In the first versions, UML was described as addressing the needs of modeling systems in an object-oriented manner
- Object orientation still is the inspiration for some key concepts
- Main concepts:
  - Object – individual unit capable of *receiving/sending messages*, processing data
  - Class – pattern giving an abstraction for a set of objects
  - Inheritance – technique for reusability and extendibility

Class Diagrams

- **A Class diagram** gives an overview of a system by showing its classes and the relationships among them.
- Class diagrams are static -- they display what interacts but not what happens when they do interact.
- Main concepts involved
  - Class - Object
  - Inheritance
  - (various kinds of) Associations

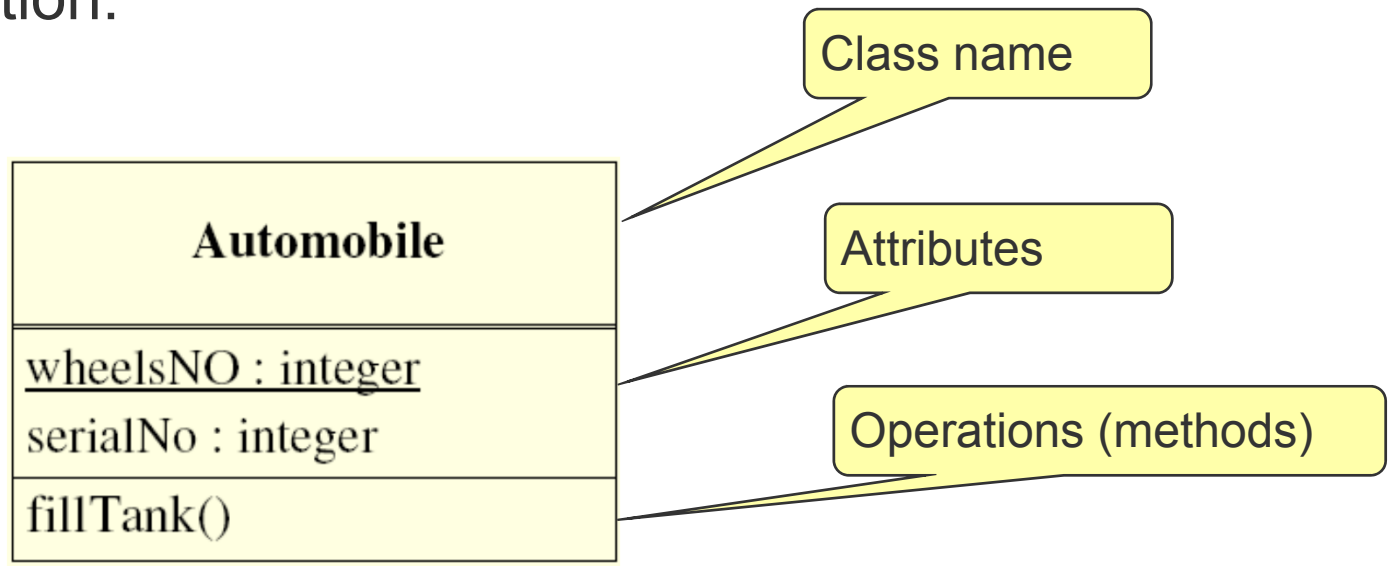
Class Diagram Example

The class diagram below models a customer order from a retail catalog. The central class is the **Order**. Associated with it are the **Customer** making the purchase and the **Payment**. A **Payment** is one of three kinds: **Cash**, **Check**, or **Credit**. The order contains **OrderDetails** (line items), each with its associated **Item**.

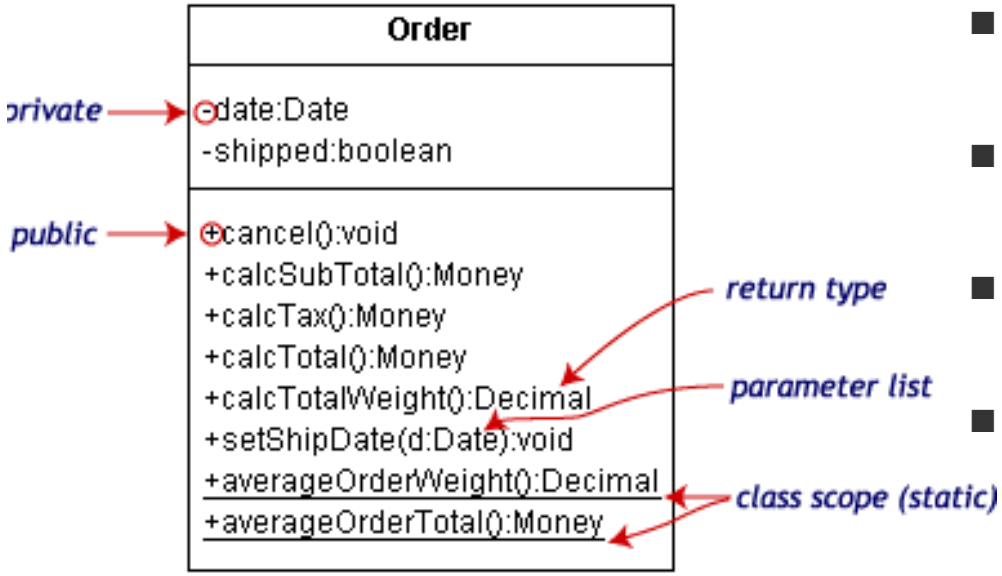


UML Class

- Gives the type of a set of objects existing at run-time
- Declares a collection of methods and attributes that describe the structure and behavior of its objects
- Basic notation:



Class Information



- UML class notation is a rectangle divided into three parts: class name, attributes, and operations.
- Names of abstract classes, such as ***Payment***, are in italics.
- Relationships between classes are the connecting links.
- Attributes and operations can be labeled according to access and scope.
- The illustration uses the following UML™ conventions.
  - Static members are underlined. Instance members are not.
  - The operations follow this form: *<access specifier> <name>* (*<parameter list>*) : *<return type>*
  - The parameter list shows each parameter type preceded by a colon.
  - Access specifiers appear in front of each member.

Access specifiers:

Symbol	Access
+	public: they are visible to all
–	private: not visible to callers outside the class
#	protected: only visible to children of the class

Class Diagram Elements

- **Association** -- a relationship between instances of the two classes. In a diagram, an association is a link connecting two classes.
- **Aggregation** -- an association in which one class belongs to a collection. An aggregation has a diamond end pointing to the part containing the whole.
  - **Order** has a collection of **OrderDetails**.
- **Generalization** -- an inheritance link indicating one class is a superclass of the other. A generalization has a triangle pointing to the superclass.
  - ***Payment*** is a superclass of **Cash**, **Check**, and **Credit**.
- An end of an assiciation may have a **role name** to clarify the nature of the association.
  - **OrderDetail** is a line item of each **Order**
- A **navigability** arrow on an association shows which direction the association can be traversed or queried. The arrow also indicates who "owns" the association's implementation
  - **OrderDetail** has an **Item**..
  - An **OrderDetail** can be queried about its **Item**, but not the other way around

  Associations with no navigability arrows are bi-directional

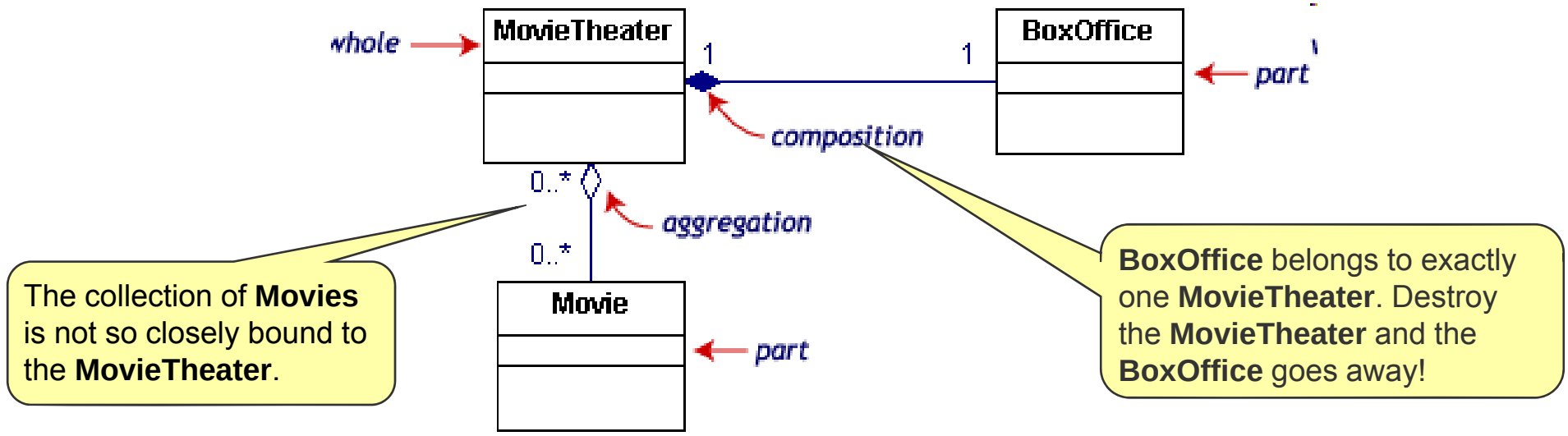
Class Diagram Elements (cont.)

- The **multiplicity** of an association end is the number of possible instances of the class associated with a single instance of the other end. Multiplicities are single numbers or ranges of numbers.
  - In our example, there can be only one **Customer** for each **Order**, but a **Customer** can have any number of **Orders**.
- This table gives the most common multiplicities.

Multiplicities	Meaning
0..1	zero or one instance. The notation ***n . . m*** indicates ***n*** to ***m*** instances.
0..* *or* *****	no limit on the number of instances (including none).
1	exactly one instance
1..*	at least one instance

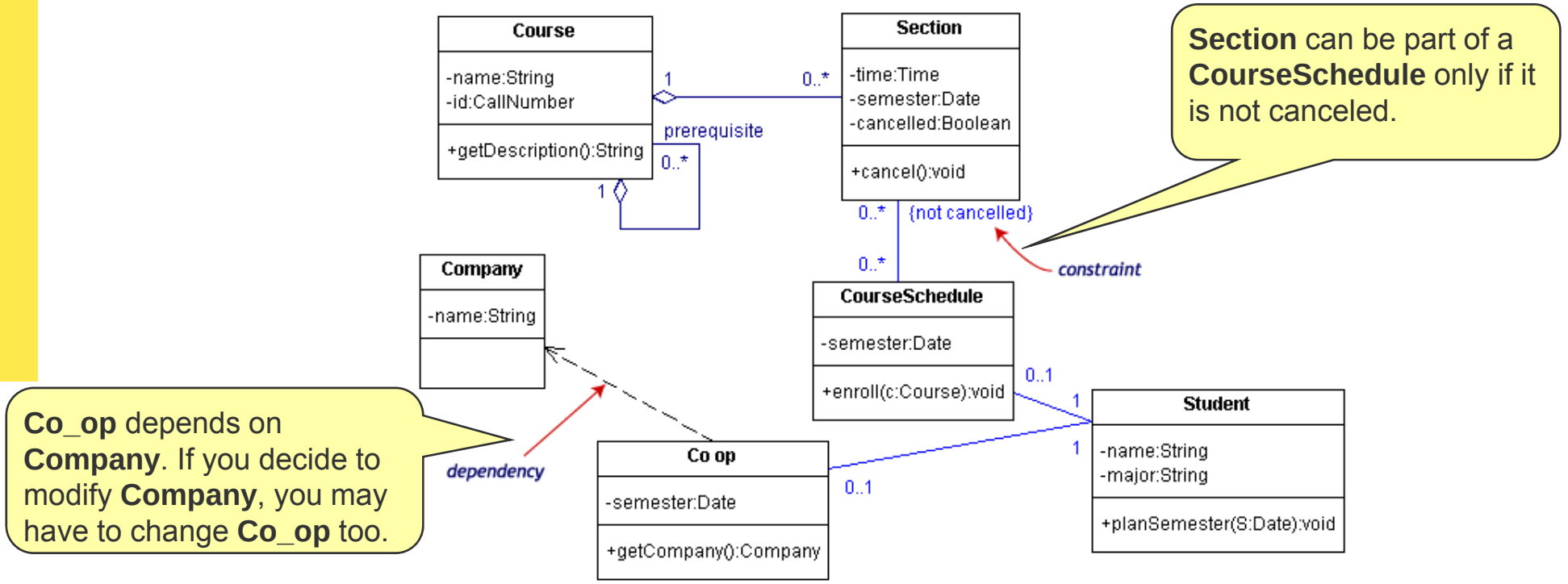
Composition and Aggregation

- **Composition** is a strong association in which the part can belong to only one whole -- the part cannot exist without the whole.
  - Composition is denoted by a filled diamond at the whole end.
- **Aggregation** is a kind of "light" composition (semantics open, to be accommodated to user needs)
  - Aggregation is denoted by a empty diamond at the whole end.



Dependencies and Constraints

- A **dependency** is a relation between two classes in which a change in one may force changes in the other. Dependencies are drawn as dotted lines.
- A **constraint** is a condition that every implementation of the design must satisfy. Constraints are written in curly braces { }.



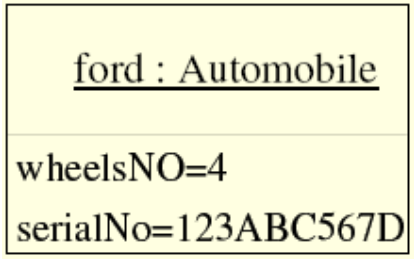
Other Elements of Class Diagrams

There are other elements of class diagrams

- Association Classes
- Interfaces
- Stereotypes
- Templates
- Comments

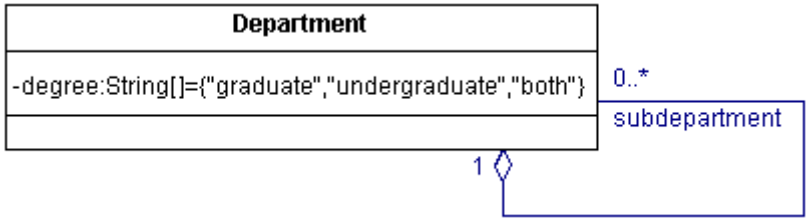
UML Object

- Instance of a class
- Can be shown in a class and object diagram
- Notation

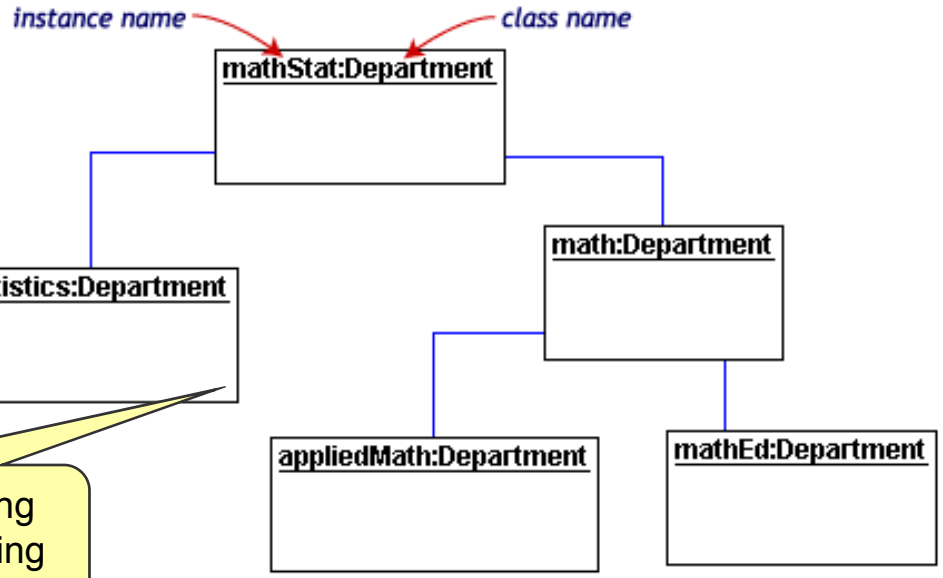


Object Diagram

- **Object diagrams** show instances instead of classes. They are useful for explaining small pieces with complicated relationships, especially recursive relationships.
- Each rectangle in the object diagram corresponds to a single instance.
- Instance names are underlined in UML diagrams.
- Class or instance names may be omitted from object diagrams as long as the diagram meaning is still clear.



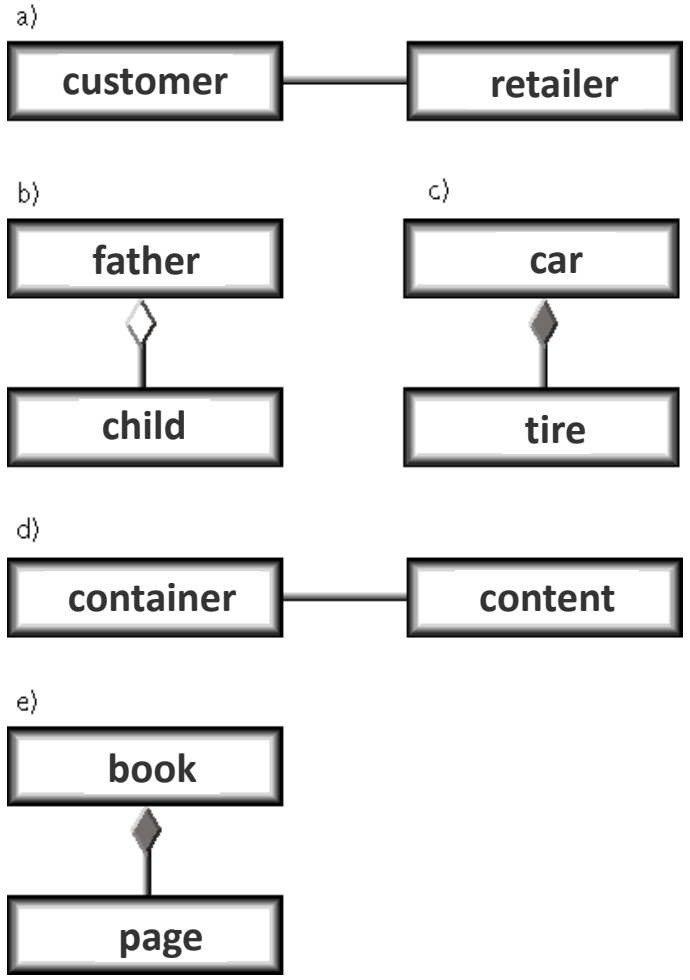
class diagram showing that a university **Department** can contain lots of other **Departments**.



object diagram instantiating the class diagram, replacing it by a concrete example.

Exercise

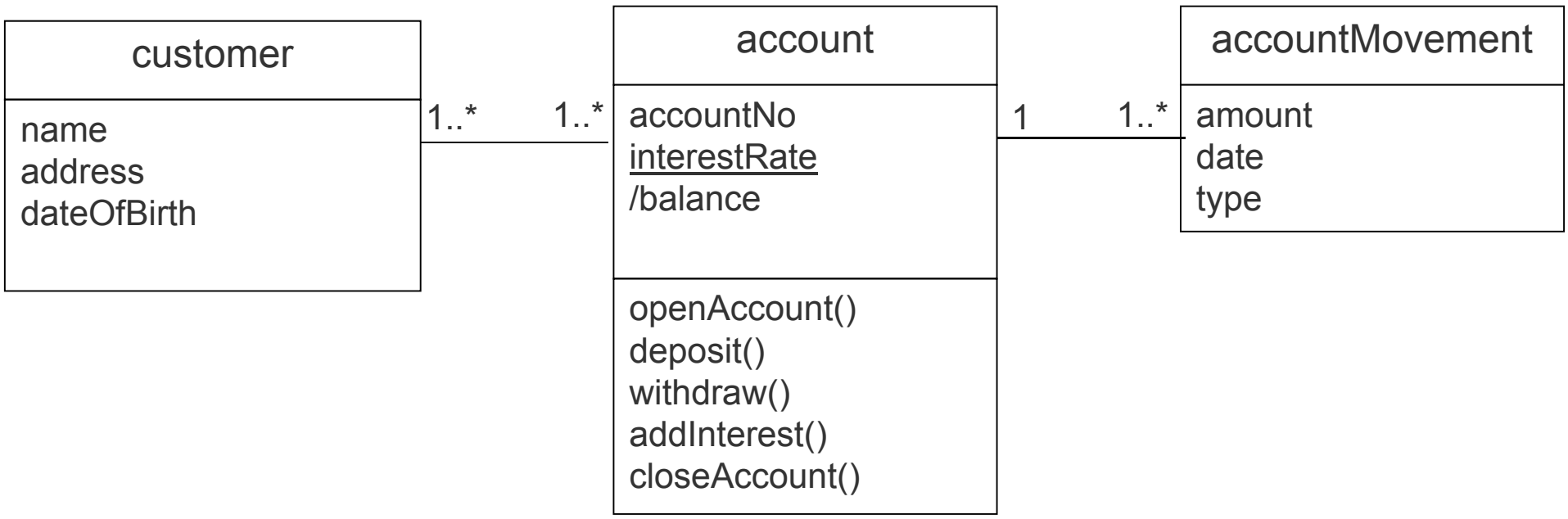
- Do the types of association (association, composition and aggregation) in die diagramms make sense?
  Give reasons for your decisions.



Exercise: Bank account

- Identify classes, attributes and operations according to the following description and draw a classs diagram.
- For the sample data draw an object diagram
  - Consider a bank and their customers. A customer can open any number of accounts. For each customer the name, address and date of birth.
  - A customer can close any of his/her accounts.
  - All accounts have a common interest rate.
  - Every account has a unique account number
  - A customer can deposit and withdraw an arbitrary amount.
  - To calculate the interest, for each account movement the date and the amount has to be noted.

Solution


customer
name
address
dateOfBirth
1..*
1..*
account
accountNo
interestRate
/balance
openAccount()
deposit()
withdraw()
addInterest()
closeAccount()
1
1..*
accountMovement
amount
date
type

Exercise

- Draw an object diagram for a customer John Smith (born 11/23/1978, living in Basel) who has an account with number 0815 who deposited 2000.- Fr. on 12/04/2008 and withdrew 500.- Fr. on 12/09/2008