UML Class Diagrams

Knut Hinkelmann

References

- OMG Unified Modelling Language - UML, Current Standard Version 2.1.2 http://www.omg.org/spec/UML/2.1.2/
- 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 (*)

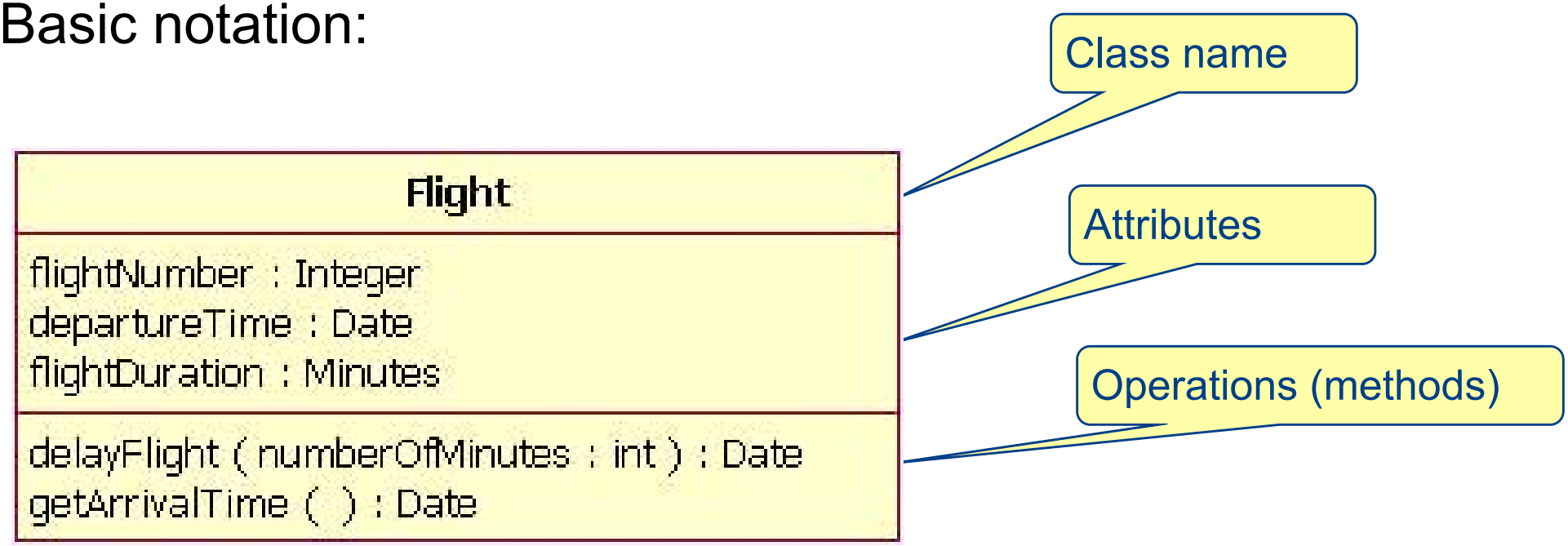
(*) not existing in UML 1.x, added in UML 2.0

Class Diagrams / Object Diagrams

- UML Class Diagrams are inspired by Object orientation (Object-oriented programming)
- Main concepts:
  - **Class** – abstraction for a set of objects with common data and operations
  - **Object** – individual unit as instance of a class
  - **Associations** – Relationship between (objects of) classes
  - **Inheritance** – technique for reusability and extendibility
- **A Class diagram** gives an overview of a system by showing its classes and the relationships among them
- An **Object diagram** additionally shows objects

UML Class

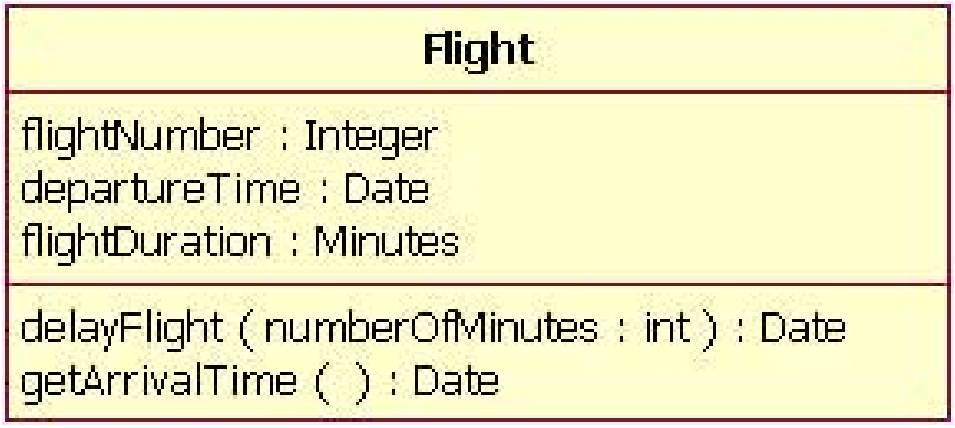
- Gives the type of a set of objects
- Declares a collection of attributes and operations (methods) that describe the structure and behavior of its objects
- Basic notation:



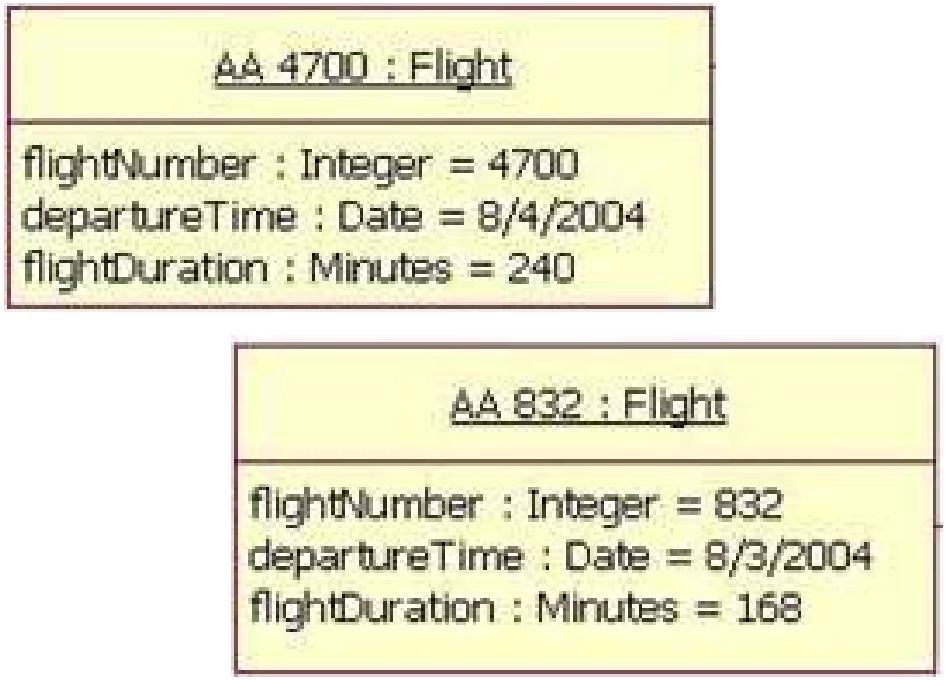
UML Class and Object

- An Object is a specific instance of a class
- It has a state which is characterized by concrete values for the attributes defined for the class

Class:

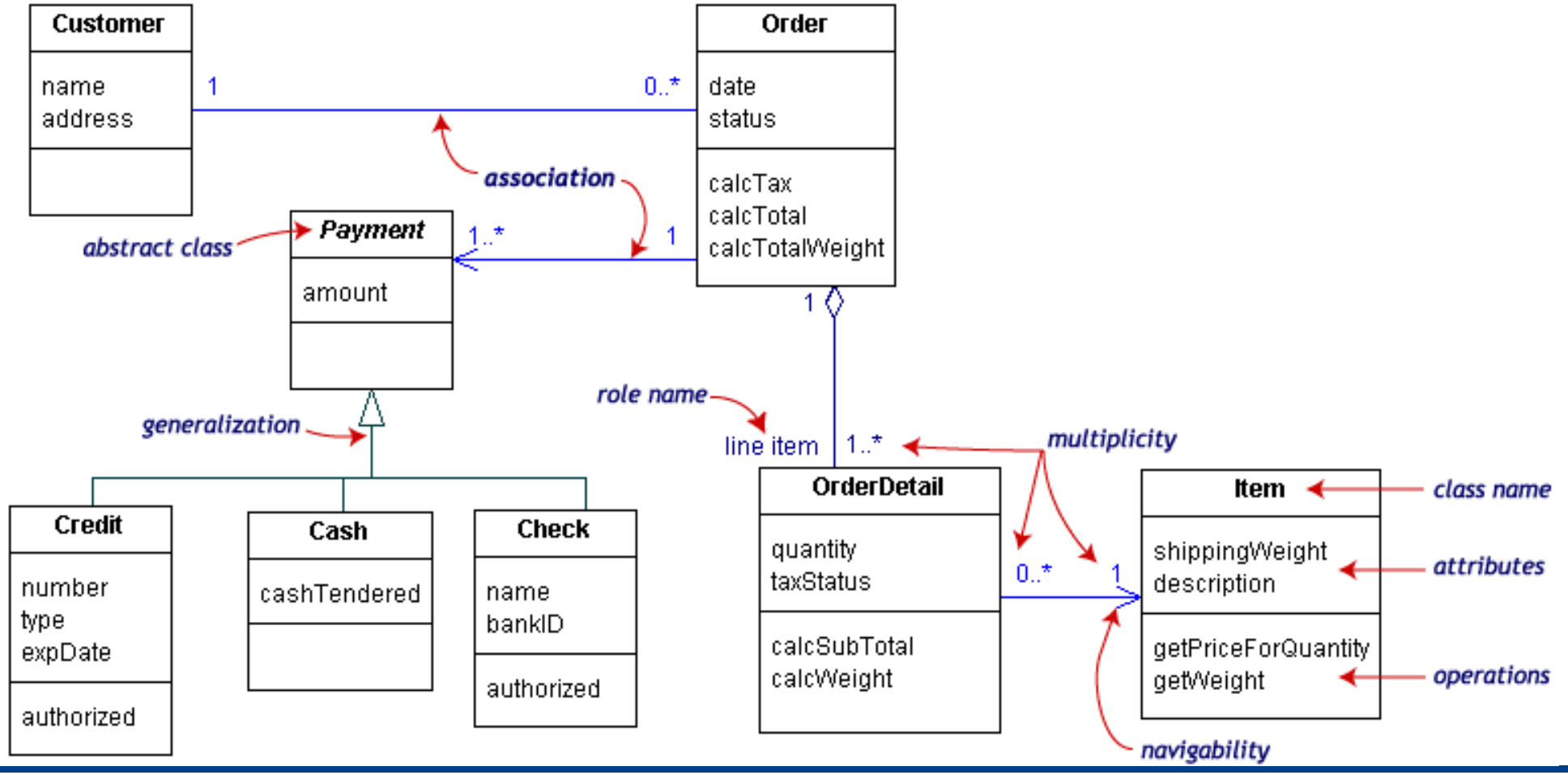


Two Objects for the class "Flight":

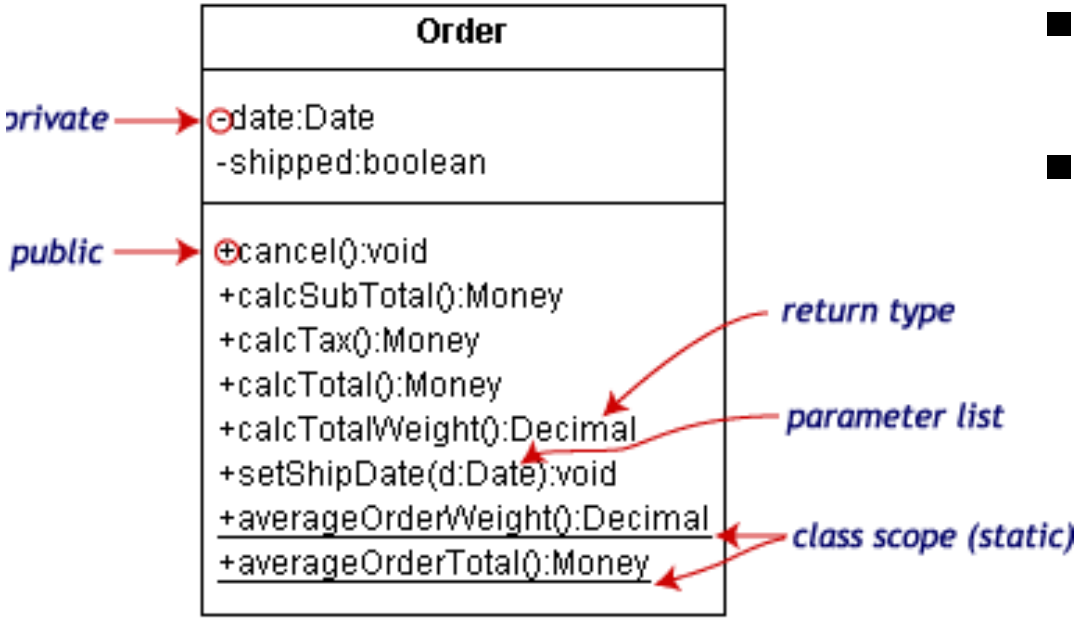


Class Diagram Example

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**.



Class Information



- UML class notation is a rectangle divided into three parts: class name, attributes, and operations.
- 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

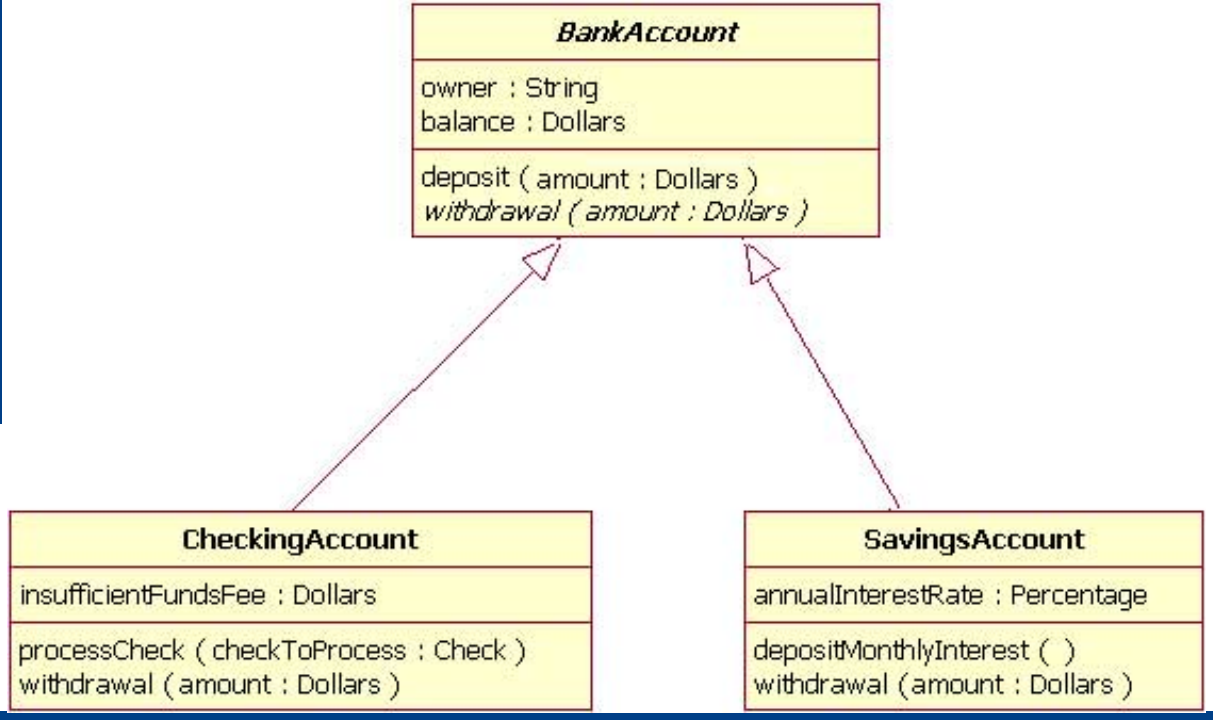
Associations

- **Association** -- a relationship between instances of the two classes. In a diagram, an association is a link connecting two classes.
- There are two special kinds of associations:
  - **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**.
  - **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**.
- 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

Generalization - Inheritance

- Inheritance is a very important concept in object-oriented design
- *Inheritance* refers to the ability of one class (child class) to *inherit* the identical functionality of another class (super class), and then add new functionality of its own.
- Inheritance is modeled with the Generalization line from the child class to the super class.



In this example, the classes CheckingAccount and SavingsAccount inherit from the BankAccount.

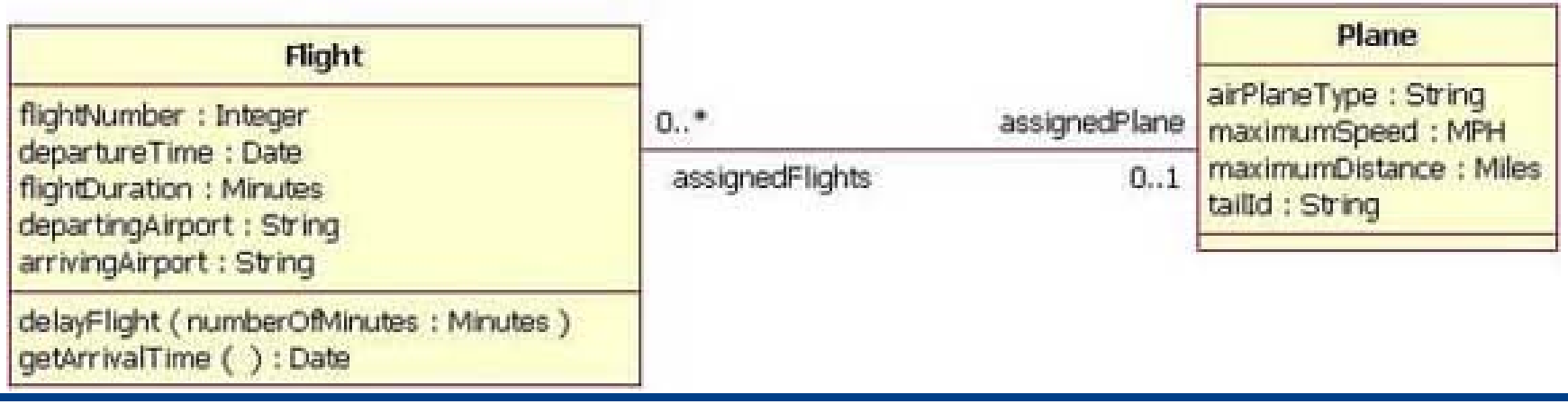
In addition to the attributes and operations explicitly mentioned, the classes CheckingAccount and SavingsAccount also have the attributes *owner* and *balance* as well as the operations *deposit()* and *withdrawal()*.

Associations

- Associations define relationships between objects
- There are five kinds of associations:
  - *Standard associations* which can be
    - bi-directional
    - uni-directional
  - *Association classes* define valuable information for associations
  - Aggregation and Composition
- Associations are always assumed to be bi-directional unless you qualify the association as some other type

Bi-directional Associations

- For bi-directional associations - indicated by a solid line between two classes - both classes are aware of each other and their relationship
- At either end of the line, you place a role name and a multiplicity value.
- This example shows that a Flight is associated with a specific Plane and a Flight. The Plane takes on the role of "assignedPlane" and the Flight the rule of "assignedFlights"

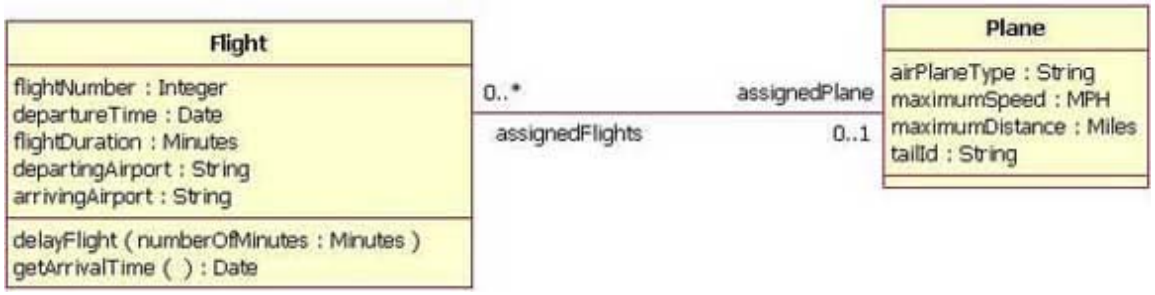


Uni-directional Association

- In a uni-directional association, two classes are related, but only one class "knows" that the relationship exists.
- A uni-directional association is drawn as a solid line with an open arrowhead pointing to the known class. Uni-directional association includes a role name and a multiplicity value, but only for the known class.

Associations vs. Attributes

- Attributes and Associations differ by the type of their value
  - Attributes have literals as values (number, string, date, …)
  - Associations have objects as their values
- Example:
  - The flight number of a Flight is an Integer
  - The plane assigned to a flight is an instance of class Plane



- Furthermore, an attribute has exactly one value, an association can have multiple values (or: there can be multiple associations)

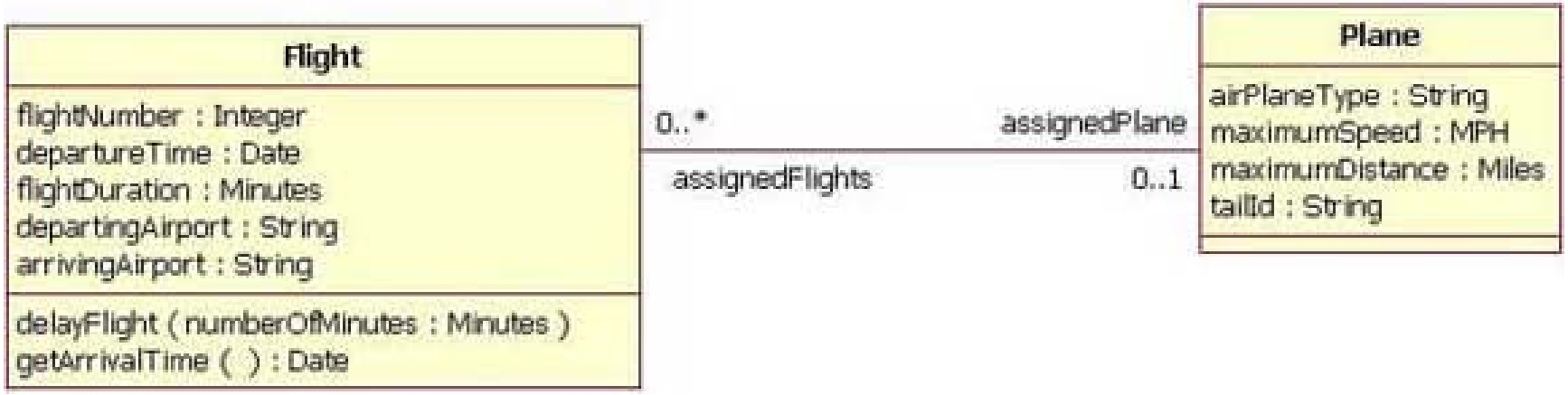
Multiplicity

- 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.
- This table gives the most common multiplicities.

Multiplicities	Meaning
0..1	zero or one instance.
0..* *or* *****	no limit on the number of instances (including none).
1	exactly one instance
1..*	at least one instance
n..m	***n*** to ***m*** instances (n and m stand for numbers, e.g. **0..4**, **3..15**)
n	exactly n instance (where n stands for a number, e.g. **3**)

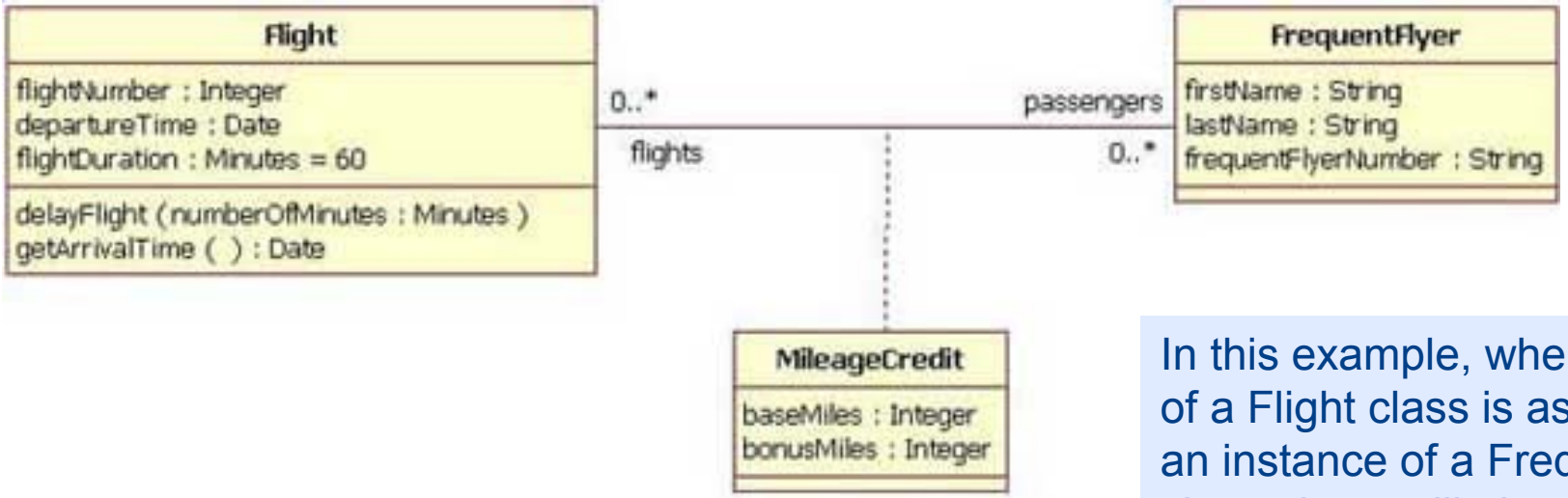
Multiplicity - Example

- The multiplicity value next to the Plane class of 0..1 means that when an instance of a Flight exists, it can either have one instance of a Plane associated with it or no Planes associated with it (i.e., maybe a plane has not yet been assigned).
- The Plane instance can be associated either with no flights or with up to an infinite number of flights.



Association Classes

- *Association class* are tied to a primary association. It includes valuable information about the relationship.
- An association class is represented like a normal class, but it is linked to an association line with a dotted line.



In this example, when an instance of a Flight class is associated with an instance of a FrequentFlyer class, there will also be an instance of a MileageCredit class

Aggregation and Compostion

- Aggregation is a special type of association used to model a "whole to its parts" relationship.
  - **In basic aggregation** relationships, the lifecycle of a *part* class is independent from the *whole* class's lifecycle.
    - Aggregation is denoted by a empty diamond at the whole end
  - For a **composition**, the child class's instance lifecycle is dependent on the parent class's instance lifecycle.
    - Composition is denoted by a filled diamond at the whole end

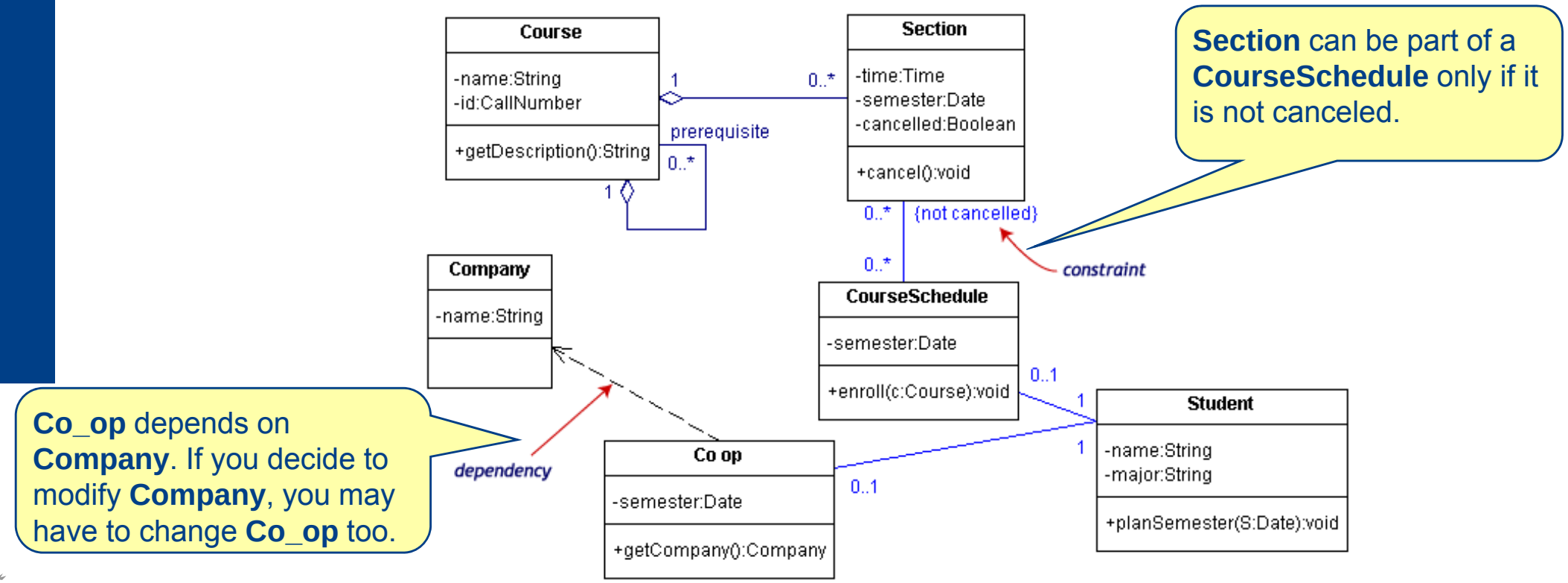
Examples of Aggregation

- Aggregation
  - The Wheel class's instance lives independently of the Car class's instance.
  - The wheel can be created before being placed on a car during assembly.
  - If the Car instance is destroyed the Wheels instance can exist further.

- Composition
  - Company class instance will always have at least one Department class instance.
  - A department cannot exist before a company exists.
  - When the Company instance is removed, the Department instance is automatically removed as well.

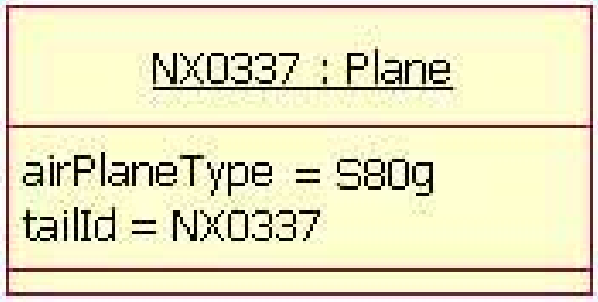
Dependencies and Constraints

- 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 { }.



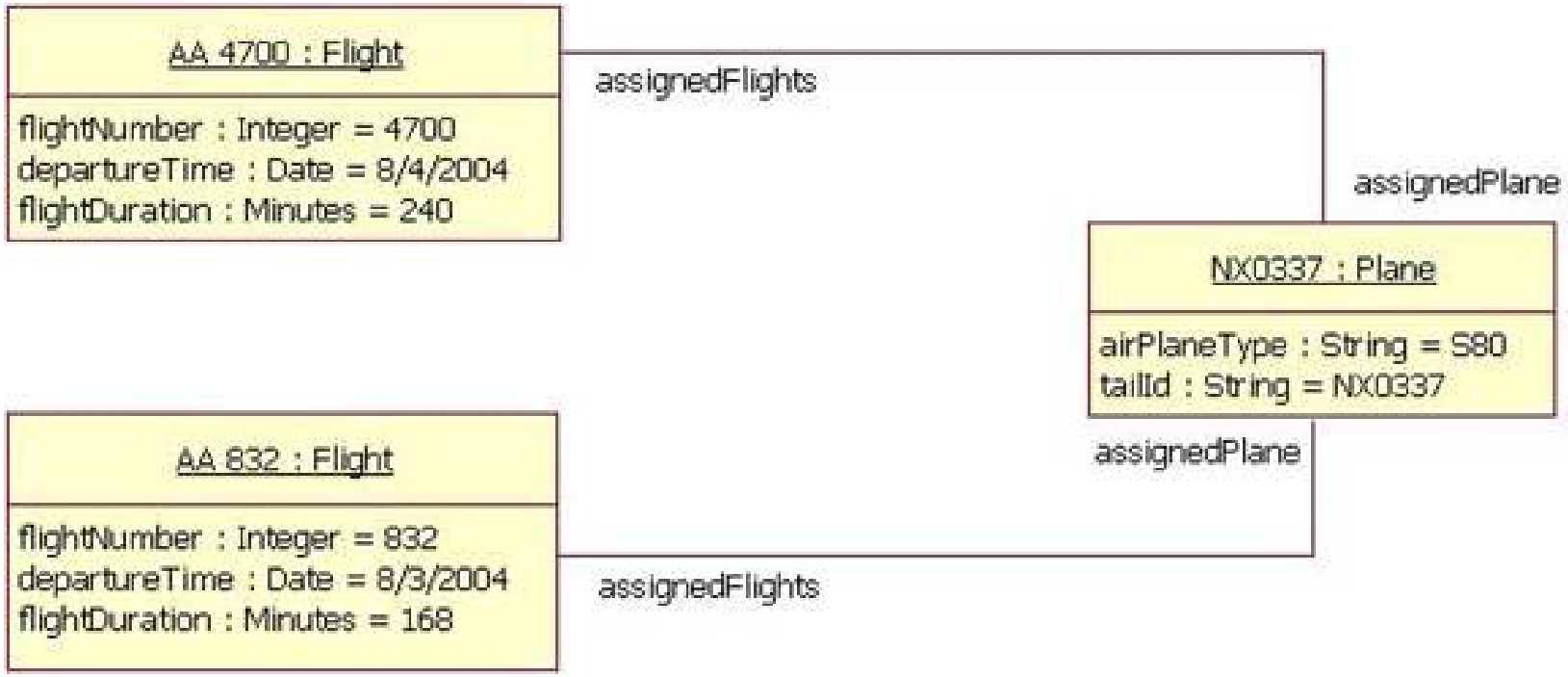
Instances - UML Objects

- Sometimes it is useful to show example instances of the classes
- The notation of an instance consists of two parts
  - The top compartment having an underlined concatenation of Instance Name and Class Name separated by a colon
  - The lower compartment having some of the attribute names and their values



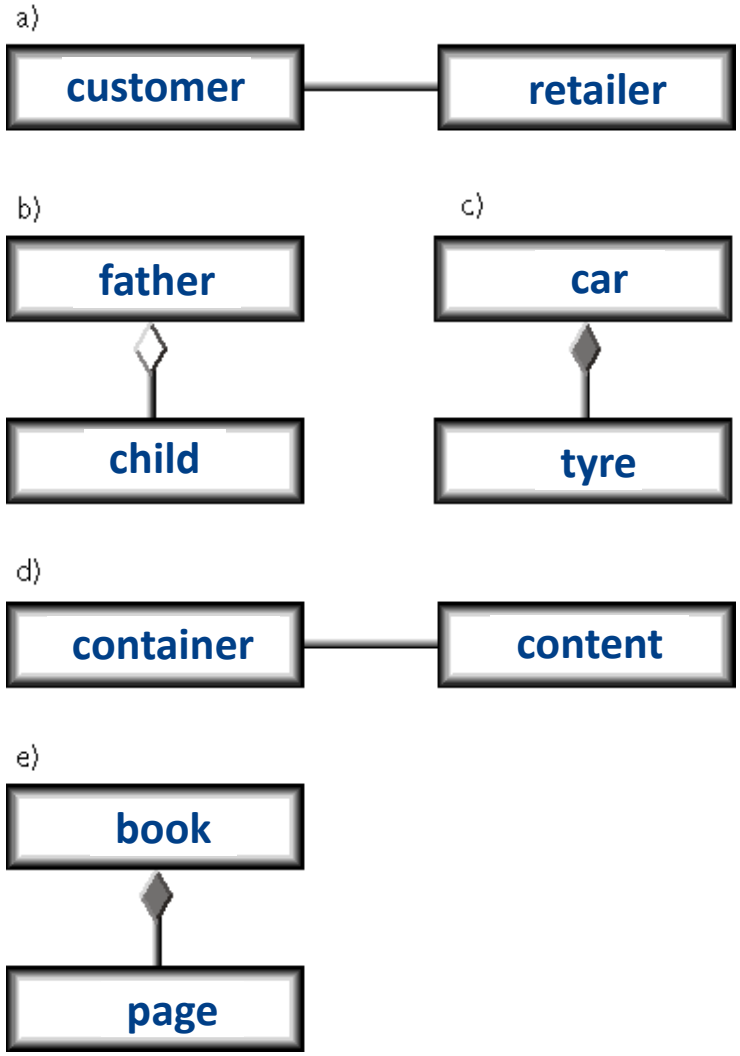
Instances

- Example: Object diagram with class instances and their associations.



Exercise

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

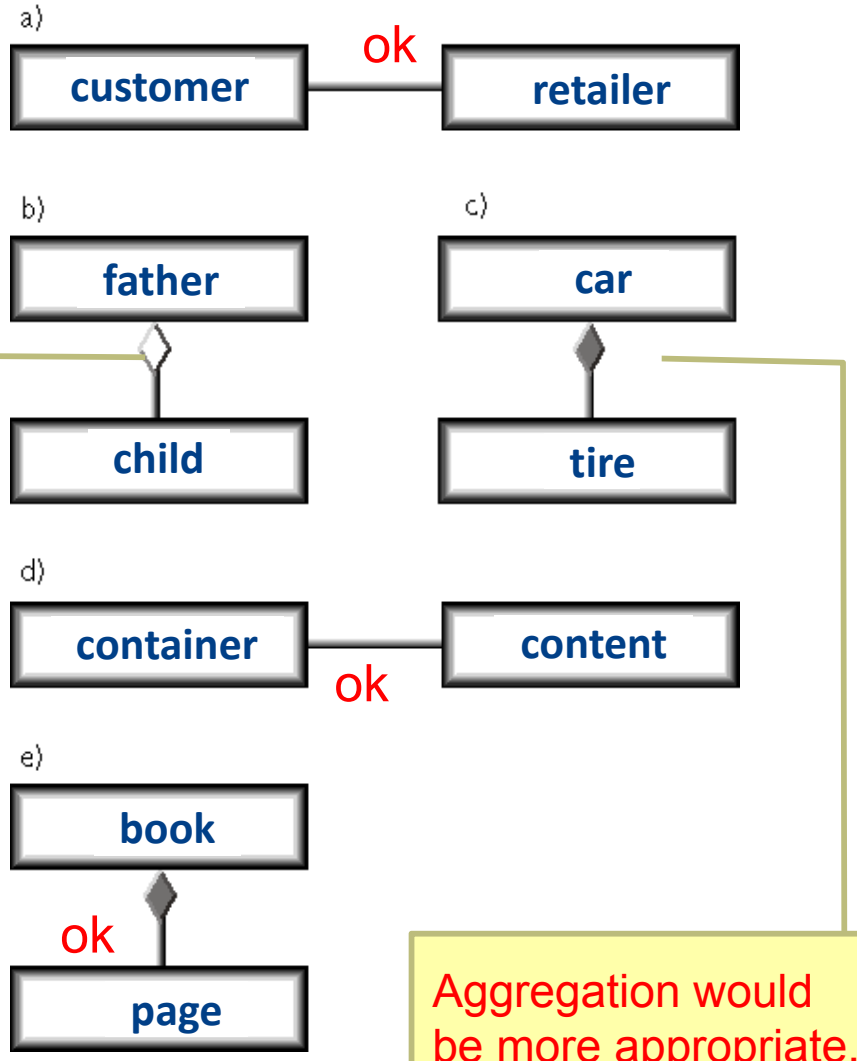


Exercise: Bank account

- 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.

Exercise

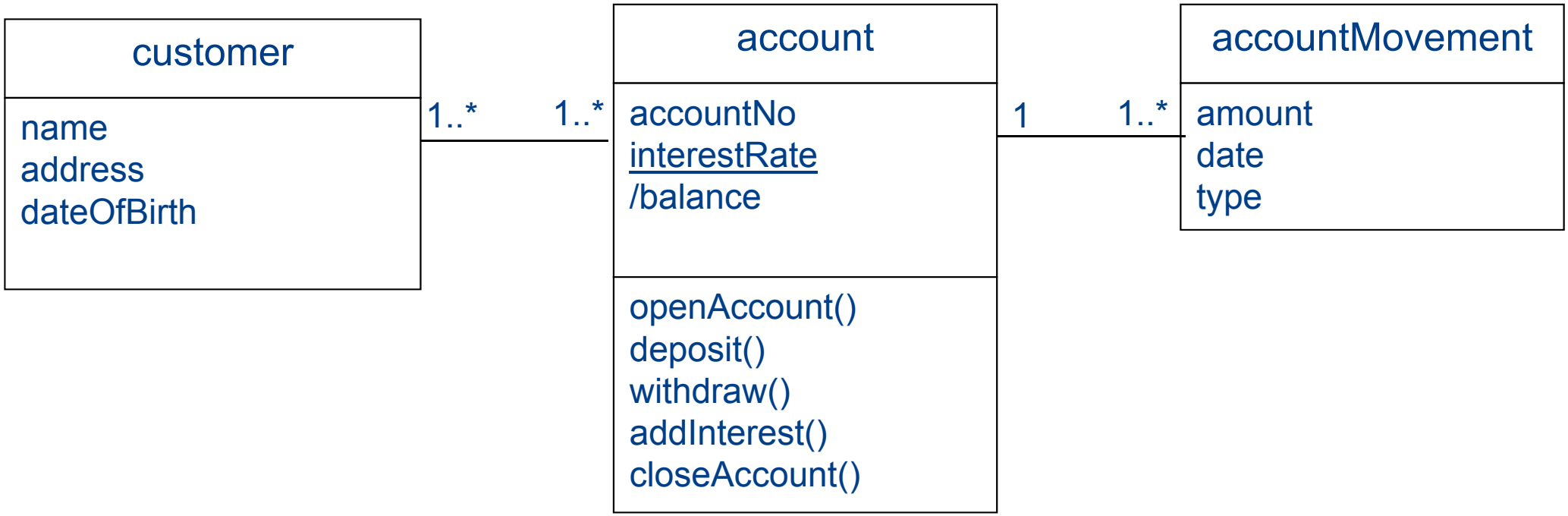
- Do the types of association (association, composition and aggregation) in die diagramms make sense?
  Give reasons for your decisions.



no, child is not part of a father but an ancestor (but part of a family)

Aggregation would be more appropriate, because tires can be separated from a car

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