UML Klassendiagramme

Inhaltsverzeichnis

1 Notation

Eine reguläre Klasse wird wie folgt dargestellt:
Oben stehen die Klassenvariablen und im unteren Teil werden die Methoden eingezeichnet. Es ist auch möglich anstatt der hier verwendeten Pascalnotation die reguläre Java Syntax zu verwenden, wie etwa für Variablen *String name* oder für Methoden *void calcHeight(int x, int y).*

ClassName
name : attribute type name : attribute type = default value
name(parameter list) : type of value returned *name(parameters list) : type of value returned*

1.1 Sichtbarkeit

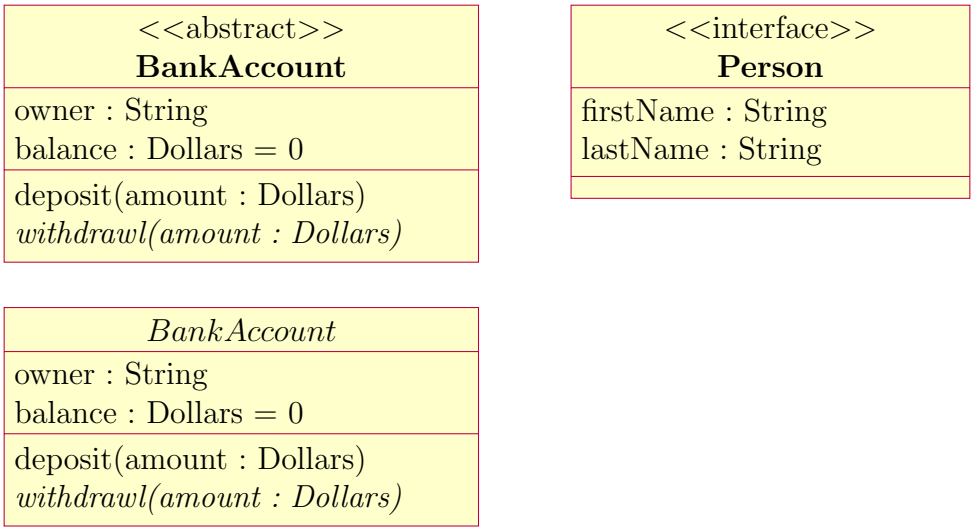
Im Regelfall werden nur die öffentlichen Schnittstellen eingezeichnet, aufgrund der Übersicht. Möchte man jedoch mehr ins Detail gehen, gibt es folgende Zeichen um die Sichtbarkeit zu kennzeichnen.

1. + . . . public
2. # . . . protected
3. entweder keine Kennzeichnung oder ∼ . . . package private
4. − . . . privat

BankAccount
+ owner : String + balance : Dollars
- deposit(amount : Dollars) ∼ withdrawal(amount : Dollars) # updateBalance(newBalance : Dollars)

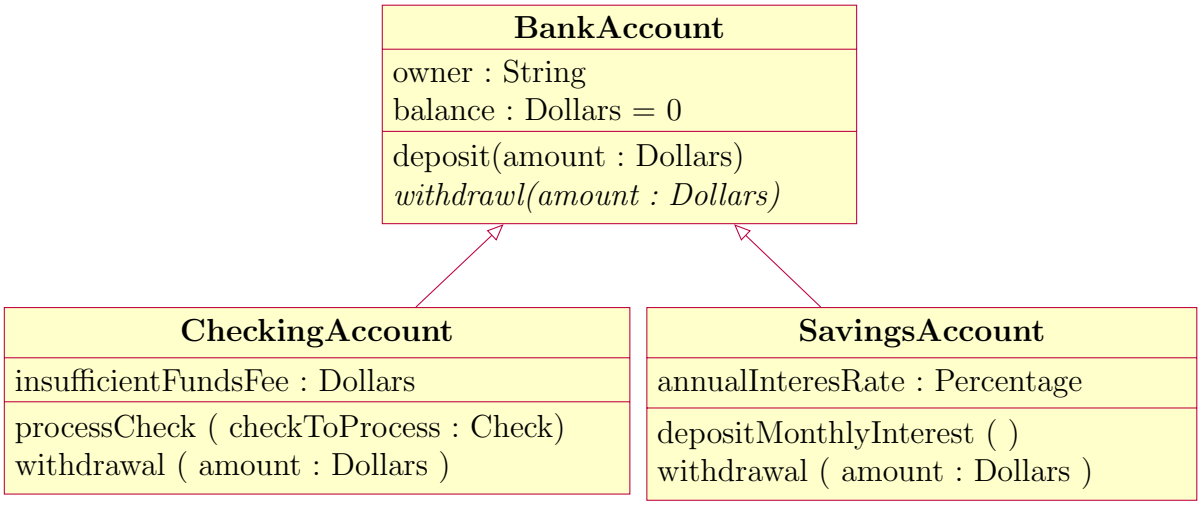
1.2 Abstrakte Klassen und Interfaces

Fr die Kennzeichnung von Interfaces und abstrakten Klassen wird <<interface>> verwendet. Bei abstrakten Klassen kann anstatt <<abstract>> einfach der Klassenname *kursiv* geschrieben werden.

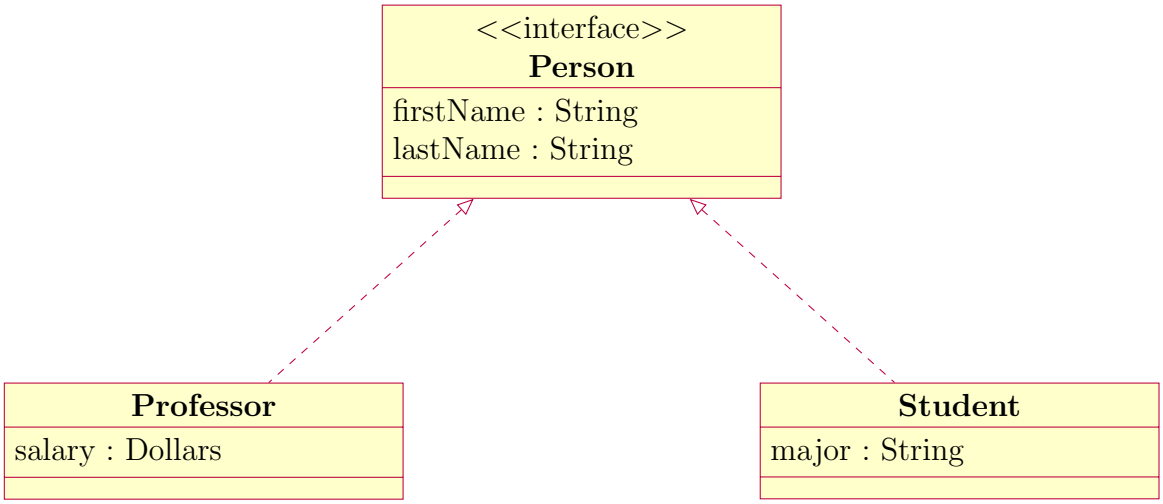


1.3 Vererbung

Eine Vererbungsbeziehung wird mittels Pfeilen mit einer durchzogenen Linie dargestellt.

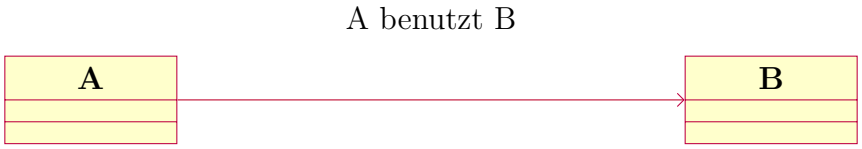


Implementierungen von **Interfaces** werden mit strichlierten Pfeilen dargestellt.

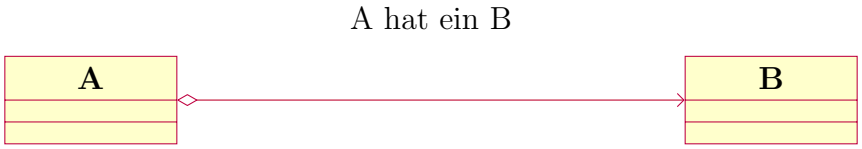


1.4 Beziehungen zwischen Klassen

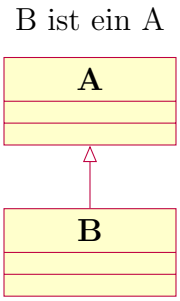
Assozation benutzt-Beziehung



Aggregation hat-Beziehung



Generalisierung ist-Beziehung

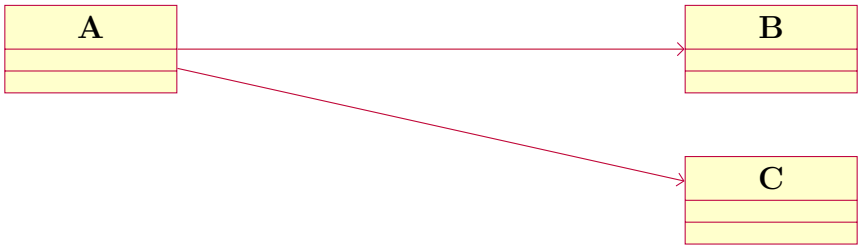


hat-Beziehung: wenn man sagen kann: „besteht aus“. A hat permanente Referenz auf B.

```
class A {
        B b;
        ...
}
```

benutzt-Beziehung: wenn man sagen kann: „steht in Verbindung mit“. A hat nur temporäre Beziehung zu B und C.

```
class A {
        void foo(B b) {
                b.bar();
                C c = ...;
        }
        ...
}
```

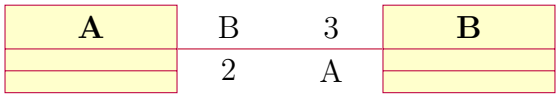


Oft wird zwischen beiden Beziehungen nicht unterschieden. Beides wird meist als benutzt-Beziehung modelliert.

1.5 Kardinalitäten von Beziehungen

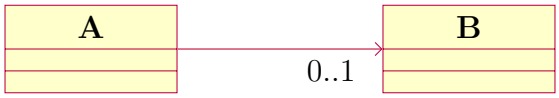
Jedes A benutzt genau 1 B

Jedes A benutzt genau 3 B und jedes B wird von 2 A benutzt



Jedes A benutzt genau 2 bis 4 B

Jedes A benutzt kann ein B oder nicht (Optinialität)

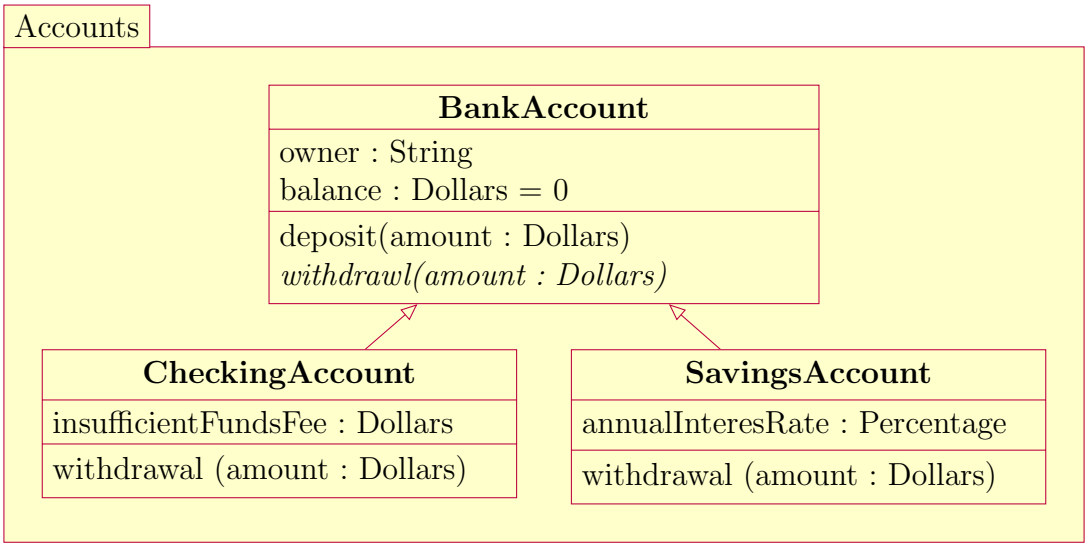


Jedes A kann null oder beliebig viele B benutzen

Die Kardinalität wird am Anfang des Modellierens oft weggelassen, außer *.

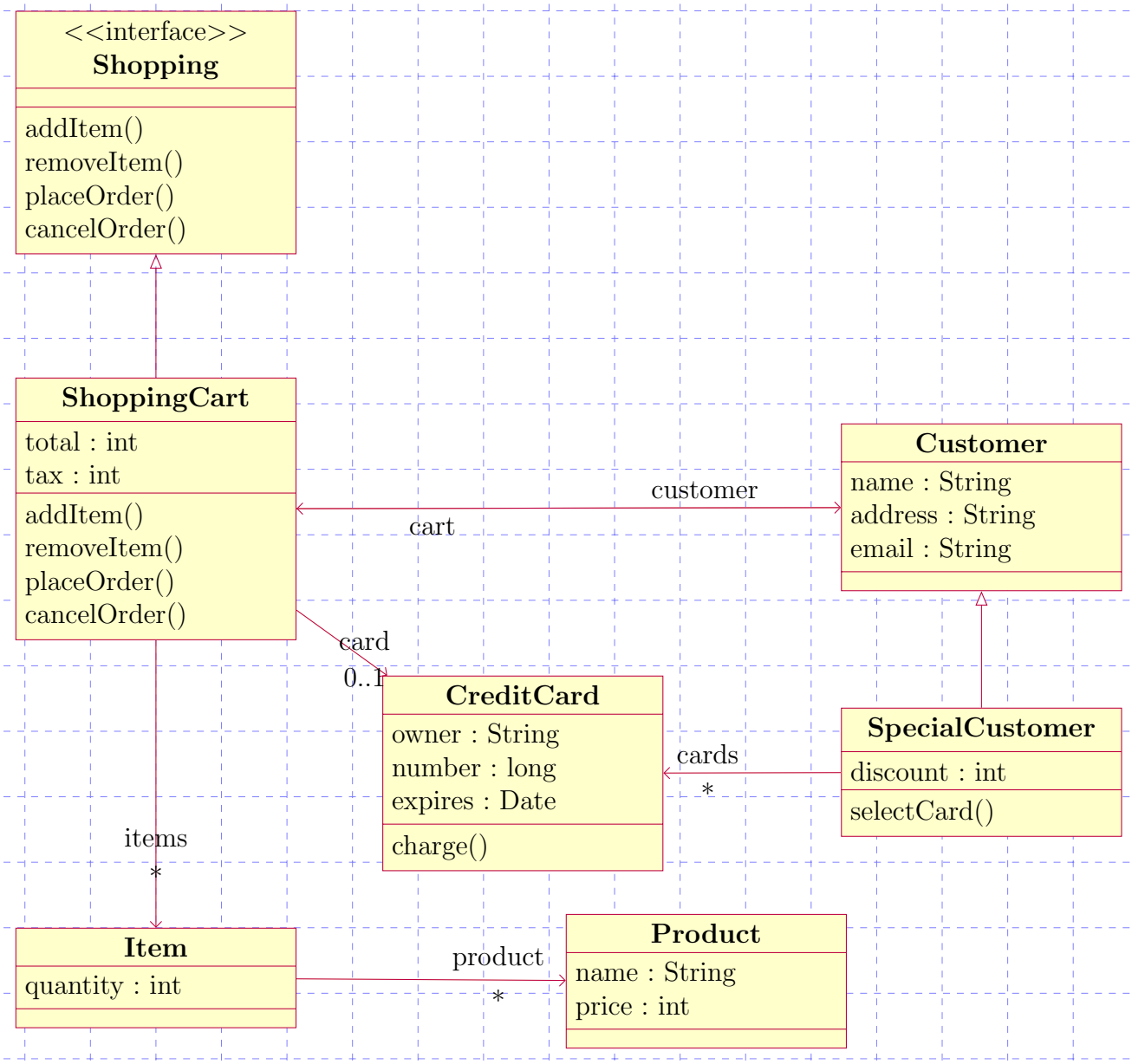
1.6 Packages

Natürlich lassen sich auch Packages darstellen. *Accounts* ist der Name des Packages.



2 Beispiel

2.1 Webshop



3 Weiterführende Links

- https://www.uml-diagrams.org/
- https://www.omg.org/spec/UML/2.5.1/PDF
- http://uml.org/
- http://uml.org/resource-hub.htm
- https://www.eclipse.org/papyrus/