Objektorientierte Programmierung

Geschichte

Warum "Objektorientierung"?

- Neue Denkweise in der Anwendungsentwicklung
  - Daten(-typen) und Funktionen so schachteln, dass diese der Realität entsprechen
  - "Sprechender" Code
- *Kostenreduzierung*
- *Erhöhung der Effizienz*

Programmiersprachen (chronologisch)

- Simula 67 (1967)
- Smalltalk (1972)
- Objective-C (1983)
- Java (1995)

[...]

Allgemeines

Objektorientierte Programmierung bezieht sich auf das Schachteln von Code innerhalb von **Klassen**.
Eine Klasse besitzt sowohl **Eigenschaften** als auch **Methoden**.
Sie ist außerdem gleichzeitig die Ausprägung eines **Datentyps**.

- **Eigenschaften** bzw. **Attribut** oder **Feld**
  beschreiben die *Daten* innerhalb eines Objekts

- **Methoden**
  beschreiben das *Verhalten* eines Objektes
  bzw. dessen *funktionalität*.
  Sie arbeiten auf objektinterne Daten.

Die Instanz einer Klasse nennt man **Objekt**.

Objekte können mittels **Nachrichten** miteinander kommunizieren.

Nachrichtentypen:

- **Accessoren**: "Gib mir einen Datensatz!"
- **Mutatoren**: "Modifiziere einen Datensatz!"
  oder: "Hol dir einen Datensatz!"

Methode vs. Funktion

Eine *Methode* kann im Gegensatz zu einer *Funktion* auf *objektinterna* zugreifen.

Das heißt: Mit einer Methode kann man, zusätzlich zur Kapselung mehrerer Anweisungen, auf *Eigenschaften* und *Methoden* des Objekts zugreifen.

Vererbung

Klassen könnnen ihr *Verhalten*, ihren *Datentyp* und ihre *Eigenschaften* an eine andere vererben.

- Bezeichnung der vererbenden Klasse: **Elternklasse** oder **Superklasse**
- Bezeichnung der erbenden Klasse: **Unterklasse** bzw. **Kindklasse**

Konstruktoren und Destruktoren

Spezialmethoden, die bei **Initialisierung** eines Objekts beziehungsweise **Deinitialisierung** eines Objekts gerufen werden.

In Swift: `init()` und `deinit()`

Sichtbarkeiten

Die **Sichtbarkeit** oder der **Scope** einer *Eigenschaft* bzw. *Methode* legt fest, aus welchem *Kontext* heraus diese zu erreichen ist.

- **Public**: Die *Eigenschaft/Methode* kann von *überall* aus angesprochen und manipuliert werden
- **Internal**: Die *Eigenschaft/Methode* kann von *allen Dateien innerhalb eines Moduls*[1] aus angesprochen und manipuliert werden
- **Private**: Die *Eigenschaft/Methode* kann *nur in der akutellen Datei* angesprochen und manipuliert werden

---

[1] The Swift Programming Language (Swift 2.1) - Access Control

Klasse vs. Instanz

- Eine *Klassenvariable/Klassenmethode* beeinflusst **alle** Objekte
- Wird eine Klassenvariable geändert, so ändert sich diese für *alle* Objekte!
- Eine *Instanzvariable/Instanzmethode* beeinflusst nur das aktuelle Objekt

Klassenvariablen bzw. *-methoden* werden mit dem Schlüsselwort **`static`** versehen.

Verträge

Spezielle Klasse:

- Enthällt ausschließlich:
  - *Methodensignaturen + Sichtbarkeit*
  - *Datentyp*

Dient dazu, *Implementierungen* eines Vertrages bestimmte Gegebenheiten erfüllen zu lassen (z.B. das Vorhandensein bestimmter Methoden oder eines bestimmten Datentyps).

Die **Signatur** einer *Methode* umfasst ihren *Namen* und ihre *Parameter*.

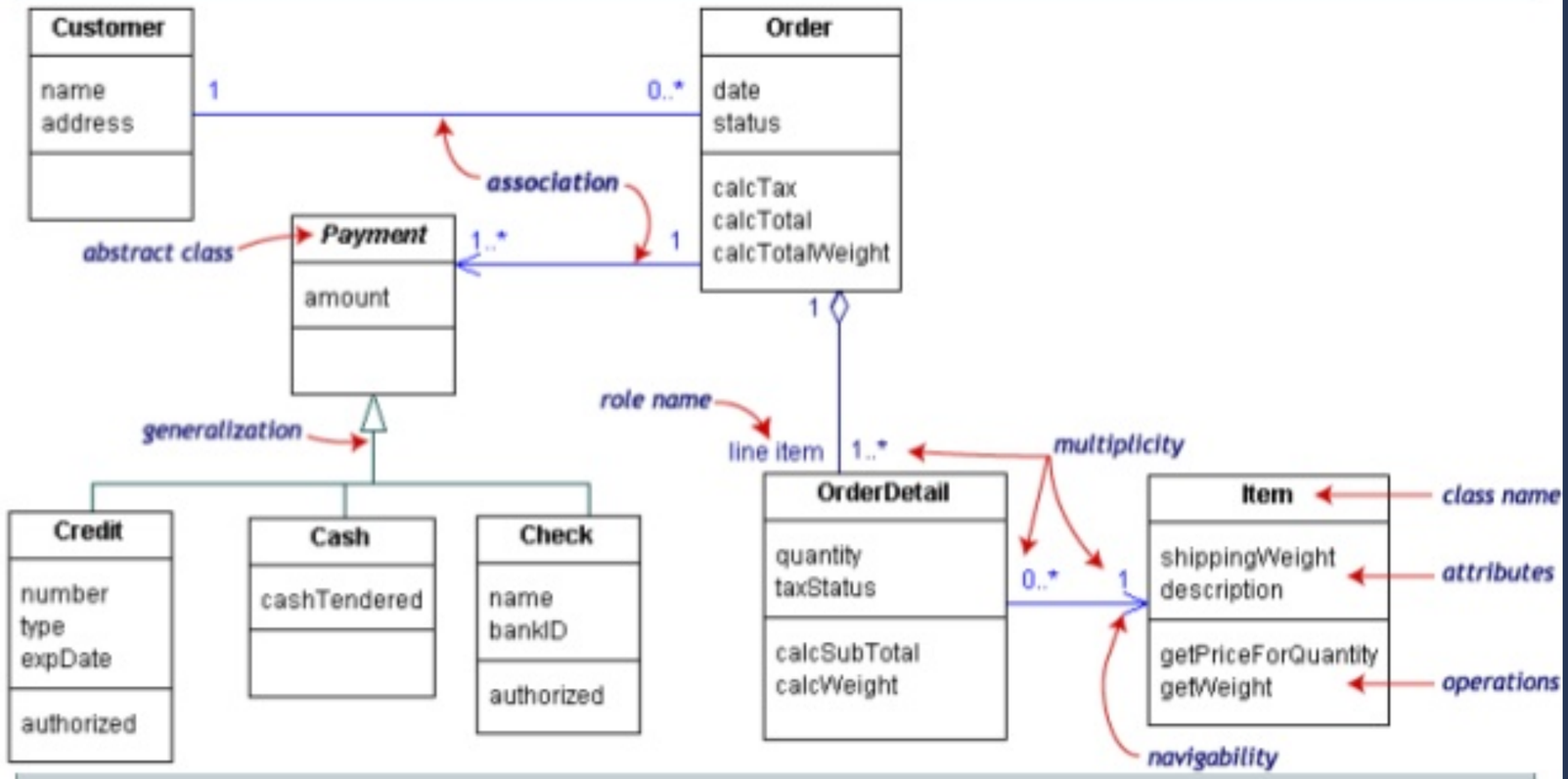
In Swift: **Protokolle**

Unified Modeling Language

Vereinheitlichung von Diagrammen, um Klassen, Objekte und deren Interaktion aufzuzeigen.

- Kompliziert
- Haarsträubend
- Basics sind gut

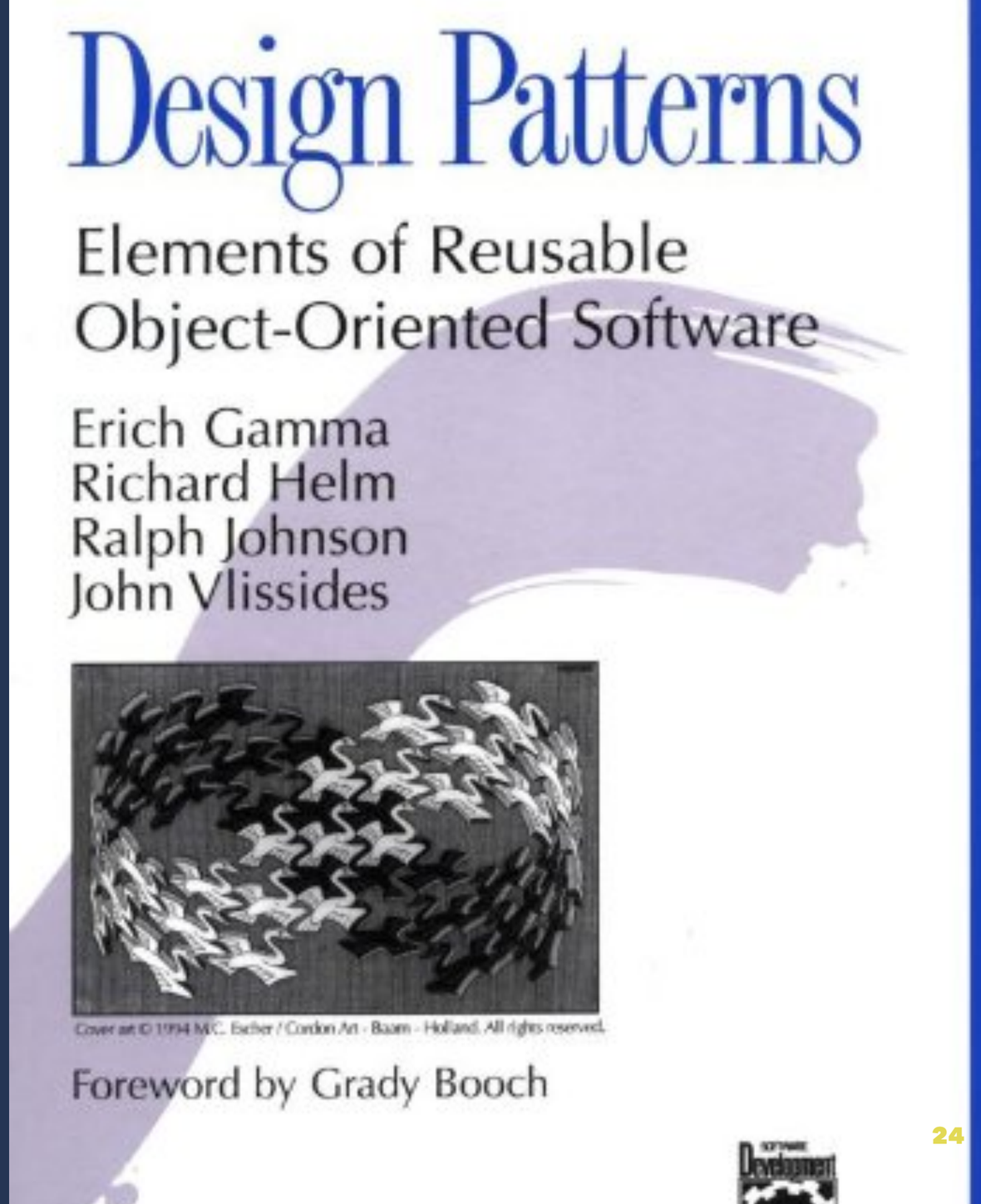
UML Class Example



Entwurfsmuster

Gegebene Klassenkonstellationen, um ein bestimmtes Szenario abzubilden.[2]

Achtung: Pattern bitte nur als Inspiration benutzen!



[2] Design Patterns - Elements of Reusable Object-Oriented Software

Fragen?

Gerne beantworte ich offene Fragen in Slack[3] und in Skype[4].

[3] SwiftDE Community in Slack

[4] Mein Skype Nickname: *osh-fan*

Demo